

GigaDevice Semiconductor Inc.

GD32VW55x
RISC-V 32-bit MCU

Firmware Library
User Guide

Revision 1.3

(Feb. 2026)

Table of Contents

Table of Contents	2
List of Figures	5
List of Tables	6
1. Introduction	24
1.1. Rules of User Manual and Firmware Library	24
1.1.1. Peripherals.....	24
1.1.2. Naming rules.....	25
2. Firmware Library Overview.....	26
2.1. File Structure of Firmware Library	26
2.1.1. Docs Folder.....	27
2.1.2. Examples Folder	28
2.1.3. Firmware Folder.....	28
2.1.4. Template Folder	29
2.1.5. Utilities Folder	32
2.2. File descriptions of Firmware Library	32
3. Firmware Library of Standard Peripherals	34
3.1. Overview of Firmware Library of Standard Peripherals.....	34
3.2. ADC	34
3.2.1. Descriptions of Peripheral registers.....	34
3.2.2. Descriptions of Peripheral functions	35
3.3. CAU	60
3.3.1. Descriptions of Peripheral registers.....	61
3.3.2. Descriptions of Peripheral functions	61
3.4. CRC	88
3.4.1. Descriptions of Peripheral registers.....	89
3.4.2. Descriptions of Peripheral functions	89
3.5. DBG	93
3.5.1. Descriptions of Peripheral registers.....	93
3.5.2. Descriptions of Peripheral functions	93
3.6. DMA.....	97
3.6.1. Descriptions of Peripheral registers.....	97
3.6.2. Descriptions of Peripheral functions	98
3.7. ECLIC	123
3.7.1. Descriptions of Peripheral functions	123

3.8. EFUSE	129
3.8.1. Descriptions of Peripheral registers	129
3.8.2. Descriptions of Peripheral functions	130
3.9. EXTI	145
3.9.1. Descriptions of Peripheral registers	145
3.9.2. Descriptions of Peripheral functions	146
3.10. FMC	153
3.10.1. Descriptions of Peripheral registers	153
3.10.2. Descriptions of Peripheral functions	154
3.11. FWDGT	172
3.11.1. Descriptions of Peripheral registers	172
3.11.2. Descriptions of Peripheral functions	173
3.12. GPIO	177
3.12.1. Descriptions of Peripheral registers	178
3.12.2. Descriptions of Peripheral functions	178
3.13. HAU	188
3.13.1. Descriptions of Peripheral registers	189
3.13.2. Descriptions of Peripheral functions	189
3.14. I2C	208
3.14.1. Descriptions of Peripheral registers	208
3.14.2. Descriptions of Peripheral functions	208
3.15. PKCAU	247
3.15.1. Descriptions of Peripheral registers	247
3.15.2. Descriptions of Peripheral functions	247
3.16. PMU	277
3.16.1. Descriptions of Peripheral registers	277
3.16.2. Descriptions of Peripheral functions	278
3.17. QSPI	295
3.17.1. Descriptions of Peripheral registers	295
3.17.2. Descriptions of Peripheral functions	296
3.18. RCU	311
3.18.1. Descriptions of Peripheral registers	311
3.18.2. Descriptions of Peripheral functions	312
3.19. RTC	344
3.19.1. Descriptions of Peripheral registers	344
3.19.2. Descriptions of Peripheral functions	345
3.20. SPI	374
3.20.1. Descriptions of Peripheral registers	374
3.20.2. Descriptions of Peripheral functions	374

3.21.	SYSCFG	391
3.21.1.	Descriptions of Peripheral registers	391
3.21.2.	Descriptions of Peripheral functions	391
3.22.	TIMER.....	396
3.22.1.	Descriptions of Peripheral registers	396
3.22.2.	Descriptions of Peripheral functions	397
3.23.	TRNG.....	453
3.23.1.	Descriptions of Peripheral registers	454
3.23.2.	Descriptions of Peripheral functions	454
3.24.	USART.....	459
3.24.1.	Descriptions of Peripheral registers	459
3.24.2.	Descriptions of Peripheral functions	460
3.25.	WWDGT.....	507
3.25.1.	Descriptions of Peripheral registers	507
3.25.2.	Descriptions of Peripheral functions	507
4.	Revision history	512

List of Figures

Figure 2-1. File structure of firmware library of GD32VW55x.....	27
Figure 2-2. Select peripheral example files	29
Figure 2-3. Copy the peripheral example files	30
Figure 2-4. Open the project file	30
Figure 2-5. Configure project files	31
Figure 2-6. Compile.....	32

List of Tables

Table 1-1. Peripherals	24
Table 2-1. Function descriptions of Firmware Library	32
Table 3-1. Peripheral function format of Firmware Library	34
Table 3-2. ADC Registers	34
Table 3-3. ADC firmware function.....	35
Table 3-4. Function adc_deinit.....	36
Table 3-5. Function adc_clock_config.....	37
Table 3-6. Function adc_enable	37
Table 3-7. Function adc_disable	38
Table 3-8. Function adc_dma_mode_enable	38
Table 3-9. Function adc_dma_mode_disable.....	39
Table 3-10. Function adc_dma_request_after_last_enable	39
Table 3-11. Function adc_dma_request_after_last_disable	40
Table 3-12. Function adc_discontinuous_mode_config	40
Table 3-13. Function adc_special_function_config	41
Table 3-14. Function adc_enable	42
Table 3-15. Function adc_disable.....	42
Table 3-16. Function adc_data_alignment_config.....	43
Table 3-17. Function adc_channel_length_config.....	43
Table 3-18. Function adc_routine_channel_config	44
Table 3-19. Function adc_inserted_channel_config	45
Table 3-20. Function adc_inserted_channel_offset_config.....	46
Table 3-21. Function adc_external_trigger_config.....	47
Table 3-22. Function adc_external_trigger_source_config	48
Table 3-23. Function adc_software_trigger_enable	49
Table 3-24. Function adc_end_of_conversion_config.....	50
Table 3-25. adc_resolution_config	50
Table 3-26. Function adc_routine_data_read.....	51
Table 3-27. Function adc_inserted_data_read	51
Table 3-28. Function adc_watchdog_single_channel_enable.....	52
Table 3-29. Function adc_watchdog_group_channel_enable	53
Table 3-30. Function adc_watchdog_disable	53
Table 3-31. Function adc_watchdog_threshold_config	54
Table 3-32. Function adc_oversample_mode_config	54
Table 3-33. Function adc_oversample_mode_enable	56
Table 3-34. Function adc_oversample_mode_disable	56
Table 3-35. Function adc_flag_get	57
Table 3-36. Function adc_flag_clear	57
Table 3-37. Function adc_interrupt_enable	58
Table 3-38. Function adc_interrupt_disable	59



Table 3-39. Function <code>adc_interrupt_flag_get</code>	59
Table 3-40. Function <code>adc_interrupt_flag_clear</code>	60
Table 3-41. CAU Registers	61
Table 3-42. CAU firmware function	61
Table 3-43. Structure <code>cau_key_parameter_struct</code>	62
Table 3-44. Structure <code>cau_iv_parameter_struct</code>	63
Table 3-45. Structure <code>cau_context_parameter_struct</code>	63
Table 3-46. Structure <code>cau_parameter_struct</code>	63
Table 3-47. Function <code>cau_deinit</code>	64
Table 3-48. Function <code>cau_struct_para_init</code>	64
Table 3-49. Function <code>cau_key_struct_para_init</code>	65
Table 3-50. Function <code>cau_iv_struct_para_init</code>	65
Table 3-51. Function <code>cau_context_struct_para_init</code>	66
Table 3-52. Function <code>cau_enable</code>	66
Table 3-53. Function <code>cau_disable</code>	67
Table 3-54. Function <code>cau_dma_enable</code>	67
Table 3-55. Function <code>cau_dma_disable</code>	68
Table 3-56. Function <code>cau_init</code>	68
Table 3-57. Function <code>cau_aes_keysize_config</code>	70
Table 3-58. Function <code>cau_key_init</code>	70
Table 3-59. Function <code>cau_iv_init</code>	71
Table 3-60. Function <code>cau_phase_config</code>	71
Table 3-61. Function <code>cau_fifo_flush</code>	72
Table 3-62. Function <code>cau_enable_state_get</code>	72
Table 3-63. Function <code>cau_data_write</code>	73
Table 3-64. Function <code>cau_data_read</code>	73
Table 3-65. Function <code>cau_context_save</code>	74
Table 3-66. Function <code>cau_context_restore</code>	75
Table 3-67. Function <code>cau_aes_ecb</code>	75
Table 3-68. Function <code>cau_aes_cbc</code>	76
Table 3-69. Function <code>cau_aes_ctr</code>	77
Table 3-70. Function <code>cau_aes_cfb</code>	78
Table 3-71. Function <code>cau_aes_ofb</code>	79
Table 3-72. Function <code>cau_aes_gcm</code>	80
Table 3-73. Function <code>cau_aes_ccm</code>	81
Table 3-74. Function <code>cau_tdes_ecb</code>	83
Table 3-75. Function <code>cau_tdes_cbc</code>	83
Table 3-76. Function <code>cau_des_ecb</code>	84
Table 3-77. Function <code>cau_des_cbc</code>	85
Table 3-78. Function <code>cau_flag_get</code>	86
Table 3-79. Function <code>cau_interrupt_enable</code>	87
Table 3-80. Function <code>cau_interrupt_disable</code>	87
Table 3-81. Function <code>cau_interrupt_flag_get</code>	88
Table 3-82. CRC Registers	89



Table 3-83. CRC firmware function	89
Table 3-84. Function crc_deinit.....	89
Table 3-85. Function crc_data_register_reset.....	90
Table 3-86. Function crc_data_register_read.....	90
Table 3-87. Function crc_free_data_register_read.....	91
Table 3-88. Function crc_free_data_register_write	91
Table 3-89. Function crc_single_data_calculate.....	92
Table 3-90. Function crc_block_data_calculate.....	92
Table 3-91. DBG Registers.....	93
Table 3-92. DBG firmware function	93
Table 3-93. Enum dbg_periph_enum	94
Table 3-94. Function dbg_deinit	94
Table 3-95. Function dbg_id_get	94
Table 3-96. Function dbg_low_power_enable.....	95
Table 3-97. Function dbg_low_power_disable.....	96
Table 3-98. Function dbg_periph_enable.....	96
Table 3-99. Function dbg_periph_disable.....	97
Table 3-100. DMA Registers.....	97
Table 3-101. DMA firmware function	98
Table 3-102. Enum dma_channel_enum	99
Table 3-103. Enum dma_subperipheral_enum.....	99
Table 3-104. Structure dma_multi_data_parameter_struct.....	100
Table 3-105. Structure dma_single_data_parameter_struct.....	100
Table 3-106. Function dma_deinit	100
Table 3-107. Function dma_single_data_para_struct_init	101
Table 3-108. Function dma_multi_data_para_struct_init	102
Table 3-109. Function dma_single_data_mode_init.....	102
Table 3-110. Function dma_multi_data_mode_init.....	103
Table 3-111. Function dma_periph_address_config	104
Table 3-112. Function dma_memory_address_config	105
Table 3-113. Function dma_transfer_number_config	106
Table 3-114. Function dma_transfer_number_get.....	106
Table 3-115. Function dma_priority_config	107
Table 3-116. Function dma_memory_burst_beats_config	107
Table 3-117. Function dma_periph_burst_beats_config	108
Table 3-118. Function dma_memory_width_config.....	109
Table 3-119. Function dma_periph_width_config.....	110
Table 3-120. Function dma_memory_address_generation_config	110
Table 3-121. Function dma_peripheral_address_generation_config.....	111
Table 3-122. Function dma_circulation_enable	112
Table 3-123. Function dma_circulation_disable	112
Table 3-124. Function dma_channel_enable	113
Table 3-125. Function dma_channel_disable	113
Table 3-126. Function dma_transfer_direction_config.....	114



Table 3-127. Function dma_switch_buffer_mode_config	114
Table 3-128. Function dma_using_memory_get.....	115
Table 3-129. Function dma_channel_subperipheral_select	116
Table 3-130. Function dma_flow_controller_config.....	116
Table 3-131. Function dma_switch_buffer_mode_enable.....	117
Table 3-132. Function dma_switch_buffer_mode_enable.....	118
Table 3-133. Function dma_fifo_status_get.....	118
Table 3-134. Function dma_flag_get	119
Table 3-135. Function dma_flag_clear	119
Table 3-136. Function dma_interrupt_enable.....	120
Table 3-137. Function dma_interrupt_disable.....	121
Table 3-138. Function dma_interrupt_flag_get	121
Table 3-139. Function dma_interrupt_flag_clear	122
Table 3-140. ECLIC firmware function	123
Table 3-141. Enum IRQn_Type	123
Table 3-142. Function eclic_global_interrupt_enable	125
Table 3-143. Function eclic_global_interrupt_disable	126
Table 3-144. Function eclic_level_threshold_set	126
Table 3-145. Function eclic_priority_group_set	127
Table 3-146. Function eclic_irq_enable	128
Table 3-147. Function eclic_irq_disable.....	128
Table 3-148. Function eclic_system_reset.....	129
Table 3-149. EFUSE Registers	129
Table 3-150. EFUSE firmware function	130
Table 3-151. Enum efuse_flag_enum.....	131
Table 3-152. Enum efuse_clear_flag_enum	131
Table 3-153. Enum efuse_int_enum	131
Table 3-154. Enum efuse_int_flag_enum	131
Table 3-155. efuse_clear_int_flag_enum.....	131
Table 3-156. Enum efuse_reg_lock_enum	132
Table 3-157. Function efuse_read	132
Table 3-158. Function efuse_write.....	133
Table 3-159. Function efuse_boot_config.....	133
Table 3-160. Function efuse_control1_config	134
Table 3-161. Function efuse_fp_config	134
Table 3-162. Function efuse_user_control_config.....	135
Table 3-163. Function efuse_res_write.....	136
Table 3-164. Function efuse_aes_key_write.....	136
Table 3-165. Function efuse_rotpk_key_write.....	137
Table 3-166. Function efuse_user_data_write.....	137
Table 3-167. Function efuse_res_read.....	138
Table 3-168. Function efuse_aes_key_read.....	138
Table 3-169. Function efuse_rotpk_key_read	139
Table 3-170. Function efuse_puid_read	140



Table 3-171. Function efuse_huk_key_read	140
Table 3-172. Function efuse_user_data_read	141
Table 3-173. Function efuse_boot_address_get	141
Table 3-174. Function efuse_lock_config	142
Table 3-175. Function efuse_flag_get	142
Table 3-176. Function efuse_flag_clear	143
Table 3-177. Function efuse_interrupt_enable	143
Table 3-178. Function efuse_interrupt_disable	144
Table 3-179. Function efuse_interrupt_flag_get	144
Table 3-180. Function efuse_interrupt_flag_clear	145
Table 3-181. EXTI Registers	145
Table 3-182. EXTI firmware function	146
Table 3-183. Enum exti_line_enum	146
Table 3-184. Enum exti_mode_enum	147
Table 3-185. Enum exti_trig_type_enum	147
Table 3-186. Function exti_deinit	147
Table 3-187. Function exti_init	148
Table 3-188. Function exti_interrupt_enable	148
Table 3-189. Function exti_interrupt_disable	149
Table 3-190. Function exti_event_enable	149
Table 3-191. Function exti_event_disable	150
Table 3-192. Function exti_software_interrupt_enable	150
Table 3-193. Function exti_software_interrupt_disable	151
Table 3-194. Function exti_flag_get	151
Table 3-195. Function exti_flag_clear	151
Table 3-196. Function exti_interrupt_flag_get	152
Table 3-197. Function exti_interrupt_flag_clear	152
Table 3-198. FMC Registers	153
Table 3-199. FMC firmware function	154
Table 3-200. fmc_state_enum	155
Table 3-201. Function fmc_unlock	155
Table 3-202. Function fmc_lock	155
Table 3-203. Function fmc_page_erase	156
Table 3-204. Function fmc_mass_erase	156
Table 3-205. Function fmc_word_program	157
Table 3-206. Function fmc_continuous_program	158
Table 3-207. Function fmc_obr_function_enable	158
Table 3-208. Function fmc_obr_function_disable	159
Table 3-209. Function ob_unlock	160
Table 3-210. Function ob_lock	160
Table 3-211. Function ob_start	161
Table 3-212. Function ob_reload	161
Table 3-213. Function ob_security_protection_config	162
Table 3-214. Function ob_user_write	162

Table 3-215. Function ob_write_protection_config	163
Table 3-216. Function fmc_no_rtdec_config	164
Table 3-217. Function fmc_offset_region_config	164
Table 3-218. Function fmc_offset_value_config	165
Table 3-219. Function fmc_wifi_trim_cal_get	166
Table 3-220. Function fmc_wifi_trim_pa_get	166
Table 3-221. Function fmc_wifi_trim_get	167
Table 3-222. Function ob_write_protection_get	167
Table 3-223. Function ob_user_get	168
Table 3-224. Function ob_security_protection_flag_get	168
Table 3-225. Function fmc_flag_get	169
Table 3-226. Function fmc_flag_clear	170
Table 3-227. Function fmc_interrupt_enable	170
Table 3-228. Function fmc_interrupt_disable	171
Table 3-229. Function fmc_interrupt_flag_get	171
Table 3-230. Function fmc_interrupt_flag_clear	172
Table 3-231. FWDGT Registers	173
Table 3-232. FWDGT firmware function	173
Table 3-233. Function fwdgt_write_enable	173
Table 3-234. Function fwdgt_write_disable	174
Table 3-235. Function fwdgt_enable	174
Table 3-236. Function fwdgt_prescaler_value_config	175
Table 3-237. Function fwdgt_reload_value_config	175
Table 3-238. Function fwdgt_counter_reload	176
Table 3-239. Function fwdgt_config	176
Table 3-240. Function fwdgt_flag_get	177
Table 3-241. GPIO Registers	178
Table 3-242. GPIO firmware function	178
Table 3-243. Function gpio_deinit	179
Table 3-244. Function gpio_mode_set	179
Table 3-245. Function gpio_output_options_set	180
Table 3-246. Function gpio_bit_set	181
Table 3-247. Function gpio_bit_reset	182
Table 3-248. Function gpio_bit_write	182
Table 3-249. Function gpio_port_write	183
Table 3-250. Function gpio_input_bit_get	183
Table 3-251. Function gpio_input_port_get	184
Table 3-252. Function gpio_output_bit_get	185
Table 3-253. Function gpio_output_port_get	185
Table 3-254. Function gpio_af_set	186
Table 3-255. Function gpio_pin_lock	187
Table 3-256. Function gpio_bit_toggle	187
Table 3-257. Function gpio_port_toggle	188
Table 3-258. HAU Registers	189

Table 3-259. HAU firmware function	189
Table 3-260. Structure hau_init_parameter_struct	190
Table 3-261. Structure hau_digest_parameter_struct	190
Table 3-262. Structure hau_context_parameter_struct	190
Table 3-263. Function hau_deinit	191
Table 3-264. Function hau_init	191
Table 3-265. Function hau_init_struct_para_init	192
Table 3-266. Function hau_reset	193
Table 3-267. Function hau_last_word_validbits_num_config	193
Table 3-268. Function hau_data_write	194
Table 3-269. Function hau_infifo_words_num_get	194
Table 3-270. Function hau_digest_read	195
Table 3-271. Function hau_digest_calculation_enable	195
Table 3-272. Function hau_multiple_single_dma_config	196
Table 3-273. Function hau_dma_enable	196
Table 3-274. Function hau_dma_disable	197
Table 3-275. Function hau_context_struct_para_init	197
Table 3-276. Function hau_context_save	198
Table 3-277. Function hau_context_restore	198
Table 3-278. Function hau_hash_sha_1	199
Table 3-279. Function hau_hmac_sha_1	199
Table 3-280. Function hau_hash_sha_224	200
Table 3-281. Function hau_hmac_sha_224	201
Table 3-282. Function hau_hash_sha_256	201
Table 3-283. Function hau_hmac_sha_256	202
Table 3-284. Function hau_hash_md5	203
Table 3-285. Function hau_hmac_md5	203
Table 3-286. Function hau_flag_get	204
Table 3-287. Function hau_flag_clear	205
Table 3-288. Function hau_interrupt_enable	205
Table 3-289. Function hau_interrupt_disable	206
Table 3-290. Function hau_interrupt_flag_get	207
Table 3-291. Function hau_interrupt_flag_clear	207
Table 3-292. I2C Registers	208
Table 3-293. I2C firmware function	208
Table 3-294. i2c_interrupt_flag_enum	210
Table 3-295. Function i2c_deinit	211
Table 3-296. Function i2c_timing_config	211
Table 3-297. Function i2c_digital_noise_filter_config	212
Table 3-298. Function i2c_analog_noise_filter_enable	213
Table 3-299. Function i2c_analog_noise_filter_disable	213
Table 3-300. Function i2c_master_clock_config	214
Table 3-301. Function i2c_master_addressing	214
Table 3-302. Function i2c_address10_header_enable	215

Table 3-303. Function i2c_address10_header_disable.....	216
Table 3-304. Function i2c_address10_enable	216
Table 3-305. Function i2c_address10_disable	217
Table 3-306. Function i2c_automatic_end_enable	217
Table 3-307. Function i2c_automatic_end_disable	218
Table 3-308. Function i2c_slave_response_to_gcall_enable	218
Table 3-309. Function i2c_slave_response_to_gcall_disable	219
Table 3-310. Function i2c_stretch_scl_low_enable	219
Table 3-311. Function i2c_stretch_scl_low_disable.....	220
Table 3-312. Function i2c_address_config	220
Table 3-313. Function i2c_address_bit_compare_config	221
Table 3-314. Function i2c_address_disable	222
Table 3-315. Function i2c_second_address_config.....	222
Table 3-316. Function i2c_second_address_disable	223
Table 3-317. Function i2c_receivied_address_get	224
Table 3-318. Function i2c_slave_byte_control_enable.....	224
Table 3-319. Function i2c_slave_byte_control_disable.....	225
Table 3-320. Function i2c_nack_enable	225
Table 3-321. Function i2c_wakeup_from_deepsleep_enable.....	226
Table 3-322. Function i2c_wakeup_from_deepsleep_disable.....	226
Table 3-323. Function i2c_enable	227
Table 3-324. Function i2c_disable	227
Table 3-325. Function i2c_start_on_bus	228
Table 3-326. Function i2c_stop_on_bus	228
Table 3-327. Function i2c_data_transmit	229
Table 3-328. Function i2c_data_receive	229
Table 3-329. Function i2c_reload_enable.....	230
Table 3-330. Function i2c_reload_disable	231
Table 3-331. Function i2c_transfer_byte_number_config.....	231
Table 3-332. Function i2c_dma_enable	232
Table 3-333. Function i2c_dma_disable	232
Table 3-334. Function i2c_pec_transfer	233
Table 3-335. Function i2c_pec_enable	233
Table 3-336. Function i2c_pec_disable	234
Table 3-337. Function i2c_pec_value_get.....	234
Table 3-338. Function i2c_smbus_alert_enable.....	235
Table 3-339. Function i2c_smbus_alert_disable.....	235
Table 3-340. Function i2c_smbus_default_addr_enable	236
Table 3-341. Function i2c_smbus_default_addr_disable	236
Table 3-342. Function i2c_smbus_host_addr_enable	237
Table 3-343. Function i2c_smbus_host_addr_disable	237
Table 3-344. Function i2c_extented_clock_timeout_enable.....	238
Table 3-345. Function i2c_extented_clock_timeout_disable.....	238
Table 3-346. Function i2c_clock_timeout_enable	239

Table 3-347. Function i2c_clock_timeout_disable	239
Table 3-348. Function i2c_bus_timeout_b_config.....	240
Table 3-349. Function i2c_bus_timeout_a_config.....	241
Table 3-350. Function i2c_idle_clock_timeout_config	241
Table 3-351. Function i2c_flag_get	242
Table 3-352. Function i2c_flag_clear	243
Table 3-353. Function i2c_interrupt_enable	243
Table 3-354. Function i2c_interrupt_disable	244
Table 3-355. Function i2c_interrupt_flag_get.....	245
Table 3-356. Function i2c_interrupt_flag_clear.....	246
Table 3-357. PKCAU Registers.....	247
Table 3-358. PKCAU firmware function	247
Table 3-359. Structure pkcau_mont_parameter_struct	248
Table 3-360. Structure pkcau_mod_parameter_struct	248
Table 3-361. Structure pkcau_mod_exp_parameter_struct.....	249
Table 3-362. Structure pkcau_arithmetic_parameter_struct	249
Table 3-363. Structure pkcau crt_parameter_struct.....	249
Table 3-364. Structure pkcau_ec_group_parameter_struct	250
Table 3-365. Structure pkcau_point_parameter_struct	250
Table 3-366. Structure pkcau_signature_parameter_struct	250
Table 3-367. Structure pkcau_hash_parameter_struct.....	251
Table 3-368. Structure pkcau_ecc_out_struct.....	251
Table 3-369. Function pkcau_deinit	251
Table 3-370. Function pkcau_mont_struct_para_init.....	251
Table 3-371. Function pkcau_mod_struct_para_init.....	252
Table 3-372. Function pkcau_mod_exp_struct_para_init	253
Table 3-373. Function pkcau_arithmetic_struct_para_init.....	253
Table 3-374. Function pkcau crt_struct_para_init.....	254
Table 3-375. Function pkcau_ec_group_struct_para_init.....	254
Table 3-376. Function pkcau_point_struct_para_init.....	255
Table 3-377. Function pkcau_signature_struct_para_init.....	255
Table 3-378. Function pkcau_hash_struct_para_init	256
Table 3-379. Function pkcau_ecc_out_struct_para_init	256
Table 3-380. Function pkcau_enable	257
Table 3-381. Function pkcau_disable	258
Table 3-382. Function pkcau_start	258
Table 3-383. Function pkcau_mode_set.....	259
Table 3-384. Function pkcau_mont_param_operation	260
Table 3-385. Function pkcau_mod_operation	261
Table 3-386. Function pkcau_mod_exp_operation	262
Table 3-387. Function pkcau_mod_inver_operation	263
Table 3-388. Function pkcau_mod_reduc_operation.....	264
Table 3-389. Function pkcau_arithmetic_operation	264
Table 3-390. Function pkcau crt_exp_operation	266

Table 3-391. Function pkcau_point_check_operation	267
Table 3-392. Function pkcau_point_mul_operation	268
Table 3-393. Function pkcau_ecdsa_sign_operation	270
Table 3-394. Function pkcau_ecdsa_verification_operation	271
Table 3-395. Function pkcau_flag_get.....	273
Table 3-396. Function pkcau_flag_clear.....	274
Table 3-397. Function pkcau_interrupt_enable.....	275
Table 3-398. Function pkcau_interrupt_disable.....	275
Table 3-399. Function pkcau_interrupt_flag_get	276
Table 3-400. Function pkcau_interrupt_flag_clear	276
Table 3-401. PMU Registers	277
Table 3-402. PMU firmware function	278
Table 3-403. Function pmu_deinit	278
Table 3-404. Function pmu_lvd_select	279
Table 3-405. Function pmu_lvd_disable.....	280
Table 3-406. Function pmu_backup_write_enable	280
Table 3-407. Function pmu_backup_write_disable	281
Table 3-408. Function pmu_to_sleepmode	281
Table 3-409. Function pmu_to_deepsleepmode	282
Table 3-410. Function pmu_to_standbymode	282
Table 3-411. Function pmu_wakeup_pin_enable	283
Table 3-412. Function pmu_wakeup_pin_disable	284
Table 3-413. Function pmu_wifi_power_enable.....	284
Table 3-414. Function pmu_wifi_power_disable.....	285
Table 3-415. Function pmu_wifi_sram_control	285
Table 3-416. Function pmu_ble_control	286
Table 3-417. Function pmu_ble_wakeup_request_enable	286
Table 3-418. Function pmu_ble_wakeup_request_disable	287
Table 3-419. Function pmu_pll_force_enable	287
Table 3-420. Function pmu_pll_force_disable	288
Table 3-421. Function pmu_ble_rf_config	288
Table 3-422. Function pmu_rf_force_enable	289
Table 3-423. Function pmu_rf_force_disable	290
Table 3-424. Function pmu_rf_sequence_config	290
Table 3-425. Function pmu_flag_get.....	291
Table 3-426. Function pmu_flag_clear.....	292
Table 3-427. Function pmu_interrupt_enable.....	293
Table 3-428. Function pmu_interrupt_disable.....	293
Table 3-429. Function pmu_interrupt_flag_get	294
Table 3-430. Function pmu_interrupt_flag_clear	294
Table 3-431. QSPI registers	295
Table 3-432. QSPI firmware function.....	296
Table 3-433. Structure qspi_init_struct	296
Table 3-434. Structure qspi_command_struct	297

Table 3-435. Structure qspi_polling_struct.....	297
Table 3-436. Function qspi_deinit	297
Table 3-437. Function qspi_struct_para_init	298
Table 3-438. Function qspi_cmd_struct_para_init	298
Table 3-439. Function qspi_polling_struct_para_init.....	299
Table 3-440. Function qspi_init.....	299
Table 3-441. Function qspi_enable	300
Table 3-442. Function qspi_disable	301
Table 3-443. Function qspi_dma_enable.....	301
Table 3-444. Function qspi_dma_disable.....	302
Table 3-444. Function qspi_data_length_config	302
Table 3-445. Function qspi_command_config	303
Table 3-446. Function qspi_polling_config.....	303
Table 3-447. Function qspi_memorymapped_config	304
Table 3-448. Function qspi_data_transmit.....	306
Table 3-449. Function qspi_data_receive.....	306
Table 3-450. Function qspi_transmission_abort	307
Table 3-451. Function qspi_flag_get	307
Table 3-452. Function qspi_flag_clear	308
Table 3-453. Function qspi_interrupt_enable	308
Table 3-454. Function qspi_interrupt_disable	309
Table 3-455. Function qspi_interrupt_flag_get	310
Table 3-456. Function qspi_interrupt_flag_clear	310
Table 3-457. RCU Registers	311
Table 3-458. RCU firmware function	312
Table 3-459. Enum rcu_periph_enum	313
Table 3-460. Enum rcu_periph_reset_enum	314
Table 3-461. Enum rcu_unit_enum.....	315
Table 3-462. Enum rcu_flag_enum.....	315
Table 3-463. Enum rcu_int_flag_enum	316
Table 3-464. Enum rcu_int_flag_clear_enum	316
Table 3-465. Enum rcu_int_enum	317
Table 3-466. Enum rcu_osci_type_enum	317
Table 3-467. Enum rcu_clock_freq_enum.....	317
Table 3-468. Function rcu_deinit	317
Table 3-469. Function rcu_irc16m_dfs_to_rf_enable.....	318
Table 3-470. Function rcu_irc16m_dfs_to_rf_disable.....	318
Table 3-471. Function rcu_periph_clock_enable	319
Table 3-472. Function rcu_periph_clock_disable	319
Table 3-473. Function rcu_periph_clock_sleep_enable	320
Table 3-474. Function rcu_fmc_clock_sleep_disable	320
Table 3-475. Function rcu_periph_reset_enable.....	321
Table 3-476. Function rcu_periph_reset_disable	321
Table 3-477. Function rcu_bkp_reset_enable	322

Table 3-478. Function rcu_bkp_reset_disable	322
Table 3-479. Function rcu_rfppl_cal_enable	323
Table 3-480. Function rcu_rfppl_cal_disable	323
Table 3-481. Function rcu_control_unit_powerup	324
Table 3-482. Function rcu_control_unit_powerdown	324
Table 3-483. Function rcu_system_clock_source_config	325
Table 3-484. Function rcu_system_clock_source_get	326
Table 3-485. Function rcu_ahb_clock_config	326
Table 3-486. Function rcu_apb1_clock_config	327
Table 3-487. Function rcu_apb2_clock_config	327
Table 3-488. Function rcu_ckout0_config	328
Table 3-489. Function rcu_ckout1_config	329
Table 3-490. Function rcu_pll_config	330
Table 3-491. Function rcu_plldigdiv_sys_config	330
Table 3-492. Function rcu_rtc_clock_config	331
Table 3-493. Function rcu_rtc_div_config	331
Table 3-494. Function rcu_trng_div_config	332
Table 3-495. Function rcu_i2c0_clock_config	332
Table 3-496. Function rcu_usart0_clock_config	333
Table 3-497. Function rcu_irc16m_div_config	334
Table 3-498. Function rcu_sdio_div_config	334
Table 3-499. Function rcu_flag_get	335
Table 3-500. Function rcu_all_reset_flag_clear	335
Table 3-501. Function rcu_interrupt_flag_get	336
Table 3-502. Function rcu_interrupt_flag_clear	336
Table 3-503. Function rcu_interrupt_enable	337
Table 3-504. Function rcu_interrupt_disable	337
Table 3-505. Function rcu_lxtal_drive_capability_config	338
Table 3-506. Function rcu_osci_stab_wait	338
Table 3-507. Function rcu_osci_on	339
Table 3-508. Function rcu_osci_off	339
Table 3-509. Function rcu_osci_bypass_mode_enable	340
Table 3-510. Function rcu_osci_bypass_mode_disable	340
Table 3-511. Function rcu_rf_hxtal_clock_monitor_enable	341
Table 3-512. Function rcu_rf_hxtal_clock_monitor_disable	342
Table 3-513. Function rcu_irc16m_adjust_value_set	342
Table 3-514. Function rcu_voltage_key_unlock	343
Table 3-515. Function rcu_deepsleep_voltage_set	343
Table 3-516. Function rcu_clock_freq_get	344
Table 3-517. RTC Registers	344
Table 3-518. RTC firmware function	345
Table 3-519. Structure rtc_parameter_struct	346
Table 3-520. Structure rtc_alarm_struct	347
Table 3-521. Structure rtc_timestamp_struct	347

Table 3-522. Structure rtc_tamper_struct	347
Table 3-523. Function rtc_deinit	348
Table 3-524. Function rtc_init.....	348
Table 3-525. Function rtc_init_mode_enter	349
Table 3-526. Function rtc_init_mode_exit.....	349
Table 3-527. Function rtc_register_sync_wait	350
Table 3-528. Function rtc_current_time_get.....	350
Table 3-529. Function rtc_subsecond_get.....	351
Table 3-530. Function rtc_alarm_config.....	352
Table 3-531. Function rtc_alarm_subsecond_config.....	352
Table 3-532. Function rtc_alarm_get.....	354
Table 3-533. Function rtc_alarm_subsecond_get	354
Table 3-534. Function rtc_alarm_enable	355
Table 3-535. Function rtc_alarm_disable	355
Table 3-536. Function rtc_timestamp_enable	356
Table 3-537. Function rtc_timestamp_disable	356
Table 3-538. Function rtc_timestamp_get.....	357
Table 3-539. Function rtc_timestamp_subsecond_get.....	357
Table 3-540. Function rtc_tamper_enable.....	358
Table 3-541. Function rtc_tamper_disable.....	358
Table 3-542. Function rtc_software_bkp_reset.....	359
Table 3-543. Function rtc_tamper_without_bkp_seset.....	360
Table 3-544. Function rtc_output_pin_select.....	360
Table 3-545. Function rtc_alter_output_config	361
Table 3-546. Function rtc_calibration_output_config	361
Table 3-547. Function rtc_hour_adjust.....	362
Table 3-548. Function rtc_second_adjust.....	363
Table 3-549. Function rtc_bypass_shadow_enable	363
Table 3-550. Function rtc_bypass_shadow_disable	364
Table 3-551. Function rtc_refclock_detection_enable.....	364
Table 3-552. Function rtc_refclock_detection_disable.....	365
Table 3-553. Function rtc_wakeup_enable	365
Table 3-554. Function rtc_wakeup_disable	366
Table 3-555. Function rtc_wakeup_clock_set	366
Table 3-556. Function rtc_wakeup_timer_set.....	367
Table 3-557. Function rtc_wakeup_timer_get	368
Table 3-558. Function rtc_smooth_calibration_config	368
Table 3-559. Function rtc_coarse_calibration_enable	369
Table 3-560. Function rtc_coarse_calibration_disable.....	370
Table 3-561. Function rtc_coarse_calibration_config	370
Table 3-562. Function rtc_flag_get.....	371
Table 3-563. Function rtc_flag_clear.....	372
Table 3-564. Function rtc_interrupt_enable.....	372
Table 3-565. Function rtc_interrupt_disable.....	373

Table 3-566. SPI registers	374
Table 3-567. SPI firmware function	374
Table 3-568. Structure spi_parameter_struct	375
Table 3-569. Function spi_deinit	375
Table 3-570. Function spi_struct_para_init	376
Table 3-571. Function spi_init	376
Table 3-572. Function spi_enable	377
Table 3-573. Function spi_disable	378
Table 3-574. Function spi_nss_output_enable	378
Table 3-575. Function spi_nss_output_disable	379
Table 3-576. Function spi_nss_internal_high	379
Table 3-577. Function spi_nss_internal_low	380
Table 3-578. Function spi_dma_enable	380
Table 3-579. Function spi_dma_disable	381
Table 3-580. Function spi_data_frame_format_config	381
Table 3-581. Function spi_data_transmit	382
Table 3-582. Function spi_data_receive	382
Table 3-583. Function spi_bidirectional_transfer_config	383
Table 3-584. Function spi_format_error_clear	383
Table 3-585. Function spi_crc_polynomial_set	384
Table 3-586. Function spi_crc_polynomial_get	384
Table 3-587. Function spi_crc_on	385
Table 3-588. Function spi_crc_off	385
Table 3-589. Function spi_crc_next	386
Table 3-590. Function spi_crc_get	386
Table 3-591. Function spi_crc_error_clear	387
Table 3-592. Function spi_ti_mode_enable	387
Table 3-593. Function spi_ti_mode_disable	388
Table 3-594. Function spi_interrupt_enable	388
Table 3-595. Function spi_interrupt_disable	389
Table 3-596. Function spi_interrupt_flag_get	389
Table 3-597. Function spi_flag_get	390
Table 3-598. SYSCFG Registers	391
Table 3-599. SYSCFG firmware function	391
Table 3-600. Enum syscfg_code_start_enum	392
Table 3-601. Function syscfg_deinit	392
Table 3-602. Function syscfg_exti_line_config	392
Table 3-603. Function syscfg_io_compensation_enable	393
Table 3-604. Function syscfg_io_compensation_disable	393
Table 3-605. Function syscfg_lock_config	394
Table 3-606. Function syscfg_io_compensation_ready_flag_get	394
Table 3-607. Function syscfg_sram_ownership_config	395
Table 3-608. Function syscfg_boot_mode_get	395
Table 3-609. TIMERx Registers	396

Table 3-610. TIMERx firmware function.....	397
Table 3-611. Structure timer_parameter_struct	399
Table 3-612. Structure timer_break_parameter_struct.....	400
Table 3-613. Structure timer_oc_parameter_struct.....	400
Table 3-614. Structure timer_ic_parameter_struct.....	400
Table 3-615. Function timer_deinit.....	401
Table 3-616. Function timer_struct_para_init.....	401
Table 3-617. Function timer_init	402
Table 3-618. Function timer_enable	403
Table 3-619. Function timer_disable	403
Table 3-620. Function timer_auto_reload_shadow_enable	404
Table 3-621. Function timer_auto_reload_shadow_disable	404
Table 3-622. Function timer_update_event_enable	405
Table 3-623. Function timer_update_event_disable	405
Table 3-624. Function timer_counter_alignment	406
Table 3-625. Function timer_counter_up_direction	407
Table 3-626. timer_counter_down_direction	407
Table 3-627. Function timer_prescaler_config.....	408
Table 3-628. Function timer_repetition_value_config	408
Table 3-629. Function timer_autoreload_value_config	409
Table 3-630. Function timer_counter_value_config.....	410
Table 3-631. Function timer_counter_read	410
Table 3-632. Function timer_prescaler_read	411
Table 3-633. Function timer_single_pulse_mode_config	411
Table 3-634. Function timer_update_source_config.....	412
Table 3-635. Function timer_dma_enable	413
Table 3-636. Function timer_dma_disable	413
Table 3-637. Function timer_channel_dma_request_source_select.....	414
Table 3-638. Function timer_dma_transfer_config.....	415
Table 3-639. Function timer_event_software_generate.....	416
Table 3-640. Function timer_break_struct_para_init	417
Table 3-641. Function timer_break_config	418
Table 3-642. Function timer_break_enable.....	419
Table 3-643. Function timer_break_disable	419
Table 3-644. Function timer_automatic_output_enable	420
Table 3-645. Function timer_automatic_output_disable	420
Table 3-646. Function timer_primary_output_config.....	421
Table 3-647. Function timer_channel_control_shadow_config	422
Table 3-648. Function timer_channel_control_shadow_update_config.....	422
Table 3-649. Function timer_channel_output_struct_para_init	423
Table 3-650. Function timer_channel_output_config	423
Table 3-651. Function timer_channel_output_mode_config	424
Table 3-652. Function timer_channel_output_pulse_value_config.....	426
Table 3-653. Function timer_channel_output_shadow_config	426

Table 3-654. Function timer_channel_output_fast_config.....	427
Table 3-655. Function timer_channel_output_clear_config	428
Table 3-656. Function timer_channel_output_polarity_config.....	429
Table 3-657. Function timer_channel_complementary_output_polarity_config	430
Table 3-658. Function timer_channel_output_state_config	430
Table 3-659. Function timer_channel_complementary_output_state_config	431
Table 3-660. Function timer_channel_input_struct_para_init.....	432
Table 3-661. Function timer_input_capture_config.....	432
Table 3-662. Function timer_channel_input_capture_prescaler_config	433
Table 3-663. Function timer_channel_capture_value_register_read	434
Table 3-664. Function timer_input_pwm_capture_config	435
Table 3-665. Function timer_hall_mode_config.....	436
Table 3-665. Function timer_channel_input_remap_config	436
Table 3-666. Function timer_input_trigger_source_select	437
Table 3-667. Function timer_master_output_trigger_source_select	438
Table 3-668. Function timer_slave_mode_select	439
Table 3-669. Function timer_master_slave_mode_config	440
Table 3-670. Function timer_external_trigger_config	441
Table 3-671. Function timer_quadrature_decoder_mode_config.....	442
Table 3-672. Function timer_internal_clock_config	443
Table 3-673. Function timer_internal_trigger_as_external_clock_config	443
Table 3-674. Function timer_external_trigger_as_external_clock_config	444
Table 3-675. Function timer_external_clock_mode0_config	445
Table 3-676. Function timer_external_clock_mode1_config	446
Table 3-677. Function timer_external_clock_mode1_disable	447
Table 3-678. Function timer_write_chxval_register_config.....	447
Table 3-679. Function timer_output_value_selection_config	448
Table 3-680. Function timer_flag_get	449
Table 3-681. Function timer_flag_clear	450
Table 3-682. Function timer_interrupt_enable	450
Table 3-683. Function timer_interrupt_disable	451
Table 3-684. Function timer_interrupt_flag_get.....	452
Table 3-685. Function timer_interrupt_flag_clear.....	453
Table 3-686 TRNG Registers	454
Table 3-687. TRNG firmware function.....	454
Table 3-688. Enum trng_flag_enum	454
Table 3-689. Enum trng_int_flag_enum.....	454
Table 3-690. Function trng_deinit.....	455
Table 3-691. Function trng_enable.....	455
Table 3-692 Function trng_disable.....	456
Table 3-693 Function trng_get_true_random_data	456
Table 3-694 Function trng_interrupt_enable	457
Table 3-695 Function trng_interrupt_disable	457
Table 3-696 Function trng_flag_get	458

Table 3-697. Function <code>trng_interrupt_flag_get</code>	458
Table 3-698. Function <code>trng_interrupt_flag_clear</code>	459
Table 3-699. USART Registers	459
Table 3-700. USART firmware function.....	460
Table 3-701. Enum <code>usart_flag_enum</code>	462
Table 3-702. Enum <code>usart_interrupt_flag_enum</code>	462
Table 3-703. Enum <code>usart_interrupt_enum</code>	463
Table 3-704. Enum <code>usart_invert_enum</code>	463
Table 3-705. Function <code>usart_deinit</code>	464
Table 3-706. Function <code>usart_baudrate_set</code>	464
Table 3-707. Function <code>usart_parity_config</code>	465
Table 3-708. Function <code>usart_word_length_set</code>	465
Table 3-709. Function <code>usart_stop_bit_set</code>	466
Table 3-710. Function <code>usart_enable</code>	467
Table 3-711. Function <code>usart_disable</code>	467
Table 3-712. Function <code>usart_transmit_config</code>	468
Table 3-713. Function <code>usart_receive_config</code>	468
Table 3-714. Function <code>usart_data_first_config</code>	469
Table 3-715. Function <code>usart_invert_config</code>	470
Table 3-716. Function <code>usart_overrun_enable</code>	471
Table 3-717. Function <code>usart_overrun_disable</code>	471
Table 3-718. Function <code>usart_oversample_config</code>	472
Table 3-719. Function <code>usart_sample_bit_config</code>	472
Table 3-720. Function <code>usart_receiver_timeout_enable</code>	473
Table 3-721. Function <code>usart_receiver_timeout_disable</code>	473
Table 3-722. Function <code>usart_receiver_timeout_threshold_config</code>	474
Table 3-723. Function <code>usart_data_transmit</code>	474
Table 3-724. Function <code>usart_data_receive</code>	475
Table 3-725. Function <code>usart_command_enable</code>	476
Table 3-726. Function <code>usart_address_config</code>	476
Table 3-727. Function <code>usart_address_detection_mode_config</code>	477
Table 3-728. Function <code>usart_mute_mode_enable</code>	477
Table 3-729. Function <code>usart_mute_mode_disable</code>	478
Table 3-730. Function <code>usart_mute_mode_wakeup_config</code>	478
Table 3-731. Function <code>usart_lin_mode_enable</code>	479
Table 3-732. Function <code>usart_lin_mode_disable</code>	480
Table 3-733. Function <code>usart_lin_break_detection_length_config</code>	480
Table 3-734. Function <code>usart_halfduplex_enable</code>	481
Table 3-735. Function <code>usart_halfduplex_disable</code>	481
Table 3-736. Function <code>usart_clock_enable</code>	482
Table 3-737. Function <code>usart_clock_disable</code>	482
Table 3-738. Function <code>usart_synchronous_clock_config</code>	483
Table 3-739. Function <code>usart_guard_time_config</code>	484
Table 3-740. Function <code>usart_smartcard_mode_enable</code>	484

Table 3-741. Function <code>usart_smartcard_mode_disable</code>	485
Table 3-742. Function <code>usart_smartcard_mode_nack_enable</code>	485
Table 3-743. Function <code>usart_smartcard_mode_nack_disable</code>	486
Table 3-744. Function <code>usart_smartcard_mode_early_nack_enable</code>	486
Table 3-745. Function <code>usart_smartcard_mode_early_nack_disable</code>	487
Table 3-746. Function <code>usart_smartcard_autoretry_config</code>	487
Table 3-747. Function <code>usart_block_length_config</code>	488
Table 3-748. Function <code>usart_irda_mode_enable</code>	488
Table 3-749. Function <code>usart_irda_mode_disable</code>	489
Table 3-750. Function <code>usart_prescaler_config</code>	489
Table 3-751. Function <code>usart_irda_lowpower_config</code>	490
Table 3-752. Function <code>usart_hardware_flow_rts_config</code>	490
Table 3-753. Function <code>usart_hardware_flow_cts_config</code>	491
Table 3-754. Function <code>usart_hardware_flow_coherence_config</code>	492
Table 3-755. Function <code>usart_rs45_driver_enable</code>	492
Table 3-756. Function <code>usart_rs45_driver_disable</code>	493
Table 3-757. Function <code>usart_driver_asserttime_config</code>	493
Table 3-758. Function <code>usart_driver_deasserttime_config</code>	494
Table 3-759. Function <code>usart_depolarity_config</code>	495
Table 3-760. Function <code>usart_dma_receive_config</code>	495
Table 3-761. Function <code>usart_dma_transmit_config</code>	496
Table 3-762. Function <code>usart_reception_error_dma_disable</code>	497
Table 3-763. Function <code>usart_reception_error_dma_enable</code>	497
Table 3-764. Function <code>usart_wakeup_enable</code>	498
Table 3-765. Function <code>usart_wakeup_disable</code>	498
Table 3-766. Function <code>usart_wakeup_mode_config</code>	499
Table 3-767. Function <code>usart_receive_fifo_enable</code>	499
Table 3-768. Function <code>usart_receive_fifo_disable</code>	500
Table 3-769. Function <code>usart_receive_fifo_counter_number</code>	500
Table 3-770. Function <code>usart_flag_get</code>	501
Table 3-771. Function <code>usart_flag_clear</code>	502
Table 3-772. Function <code>usart_interrupt_enable</code>	503
Table 3-773. Function <code>usart_interrupt_disable</code>	504
Table 3-774. Function <code>usart_interrupt_flag_get</code>	505
Table 3-775. Function <code>usart_interrupt_flag_clear</code>	506
Table 3-776. WWDGT Registers	507
Table 3-777. WWDGT firmware function	508
Table 3-778. Function <code>wwdgt_deinit</code>	508
Table 3-779. Function <code>wwdgt_enable</code>	508
Table 3-780. Function <code>wwdgt_counter_update</code>	509
Table 3-781. Function <code>wwdgt_config</code>	509
Table 3-782. Function <code>wwdgt_interrupt_enable</code>	510
Table 3-783. Function <code>wwdgt_flag_get</code>	511
Table 3-784. Function <code>wwdgt_flag_clear</code>	511

1. Introduction

This manual introduces firmware library of GD32VW55x devices which are 32-bit microcontrollers based on the RISC-V processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32VW55x devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

The driving code of each peripheral is concluded by a group of functions, which describes all the performance features of the peripheral. Users can drive a peripheral by a group of APIs (application programming interface), all the APIs are standardized about the code structure, function name and parameter names.

The commonly used firmware library includes all the functions of all the peripherals, so the code size and the execution speed may not be the optimal. For most applications, users can use the library functions directly, while for the applications which are strict with the code size and execution speed, the firmware library can be used as the reference resource of how to configure a peripheral, and users adjust the code according to actual needs.

The overall structure of the firmware library user manual is shown as below:

- Rules of user manual and firmware library;
- Firmware library overview;
- Functions and registers descriptions of firmware library.

1.1. Rules of User Manual and Firmware Library

1.1.1. Peripherals

Table 1-1. Peripherals

Peripherals	Descriptions
ADC	Analog-to-digital converter
CAU	Cryptographic acceleration unit
CRC	CRC calculation unit
DBG	Debug module
DMA	Direct memory access controller
EFUSE	Electronic fuse
EXTI	Interrupt/event controller

Peripherals	Descriptions
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
HAU	Hash acceleration unit
I2C	Inter-integrated circuit interface
PKCAU	Public key cryptographic acceleration unit
PMU	Power management unit
QSPI	Quad-SPI interface
RCU	Reset and clock unit
RTC	Real-time Clock
SPI	Serial peripheral interface
TRNG	True random number generator
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer

1.1.2. Naming rules

The firmware library naming rules are shown as below:

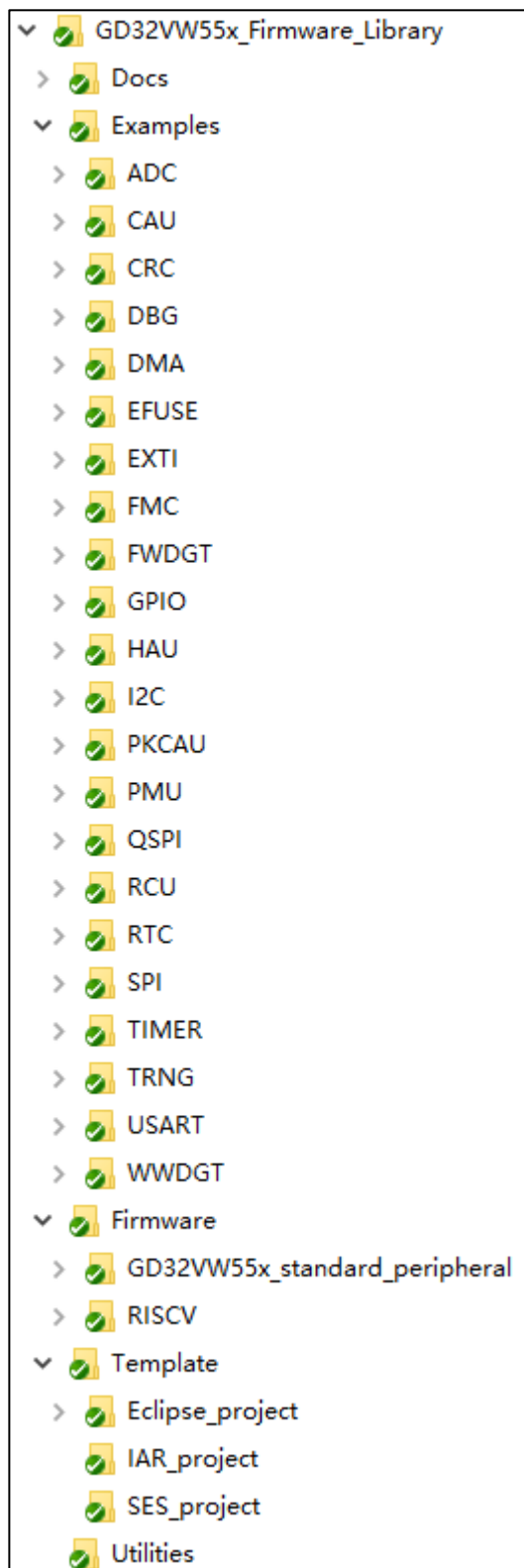
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32vw55x_”, such as: gd32vw55x_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32VW55x_Firmware_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32VW55x



2.1.1. Docs Folder

The Docs folder contains the Firmware Library User Guide and development board schematic.

2.1.2. Examples Folder

Examples folder, each of GD32 peripheral has a subfolder. Each subfolder contains one or more examples of the peripheral, to show how to use the peripheral correctly. Each of the example subfolder includes the files shown as below:

- `readme.txt`: the description and using guide of the example;
- `gd32vw55x_libopt.h`: the header file configures all the peripherals used in the example, included by different "DEFINE" sentences (all the peripherals are enabled by default);
- `gd32vw55x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32vw55x_it.h`: the header file include all the prototypes of the interrupt service routines;
- `systick.c`: the source file include the precise time delay functions by using `systick`;
- `systick.h`: the header file include the prototype of the precise time delay functions by using `systick`;
- `main.c`: example code.

Note: All the examples are not influenced by software IDEs.

2.1.3. Firmware Folder

Firmware folder includes all the subfolder and files which are the core part of the firmware:

- RISC-V subfolder:
 - Core subfolder includes the RISC-V kernel support files, users need not modify this folder;
 - `env_eclipse` subfolder includes the startup file based on the RISC-V kernel processor, exception service program and link script files of Eclipse IDE, users need not modify this folder;
 - `env_IAR` subfolder includes the startup file based on the RISC-V kernel processor and exception service program of IAR, users need not modify this folder;
 - `env_SES` subfolder includes the startup file based on the RISC-V kernel processor and exception service program of SES, users need not modify this folder;
 - `stubs` subfolder includes definition of pile functions such `_write/_read` function, users need not modify this folder.
- GD32VW55x_standard_peripheral subfolder:
 - Include subfolder includes all the header files of firmware library, users need not modify this folder;
 - Source subfolder includes all the source files of firmware library, users need not modify this folder;
 - the global header file of GD32VW55x and system configuration file, users need not modify this folder.

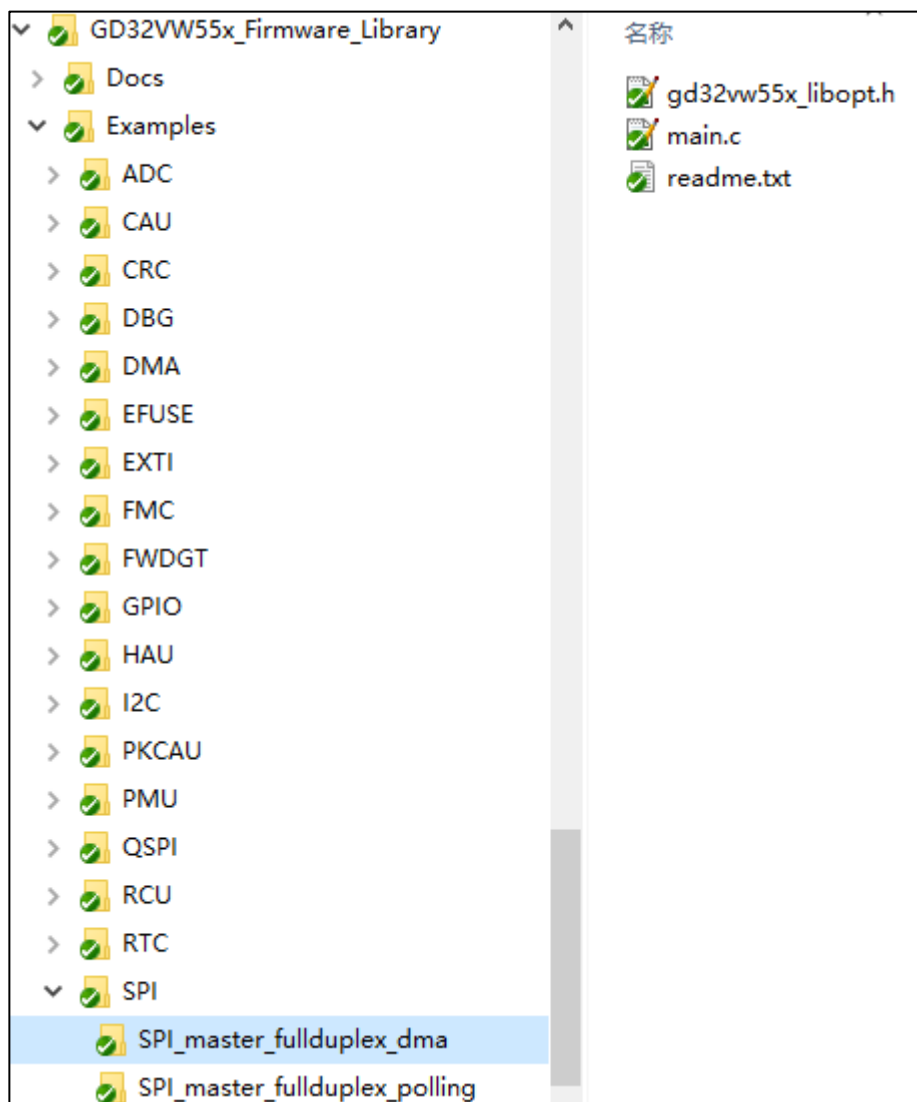
2.1.4. Template Folder

Template folder includes a simple demo of how to use LED\USART\KEY (IAR_project run in EWRISCV-3101, Eclipse_project run in GD32Eclipse, SES_project run in SEGGER Embedded Studio for RISC-V 7.32a). User can use the project template to compile the formware examples, the steps are shown as below:

Select files

Open “Examples” folder, select the module to be tested, such as SPI, open “SPI” folder, select an example of SPI, such as ” SPI_master_full duplex_dma”, shown as below:

Figure 2-2. Select peripheral example files

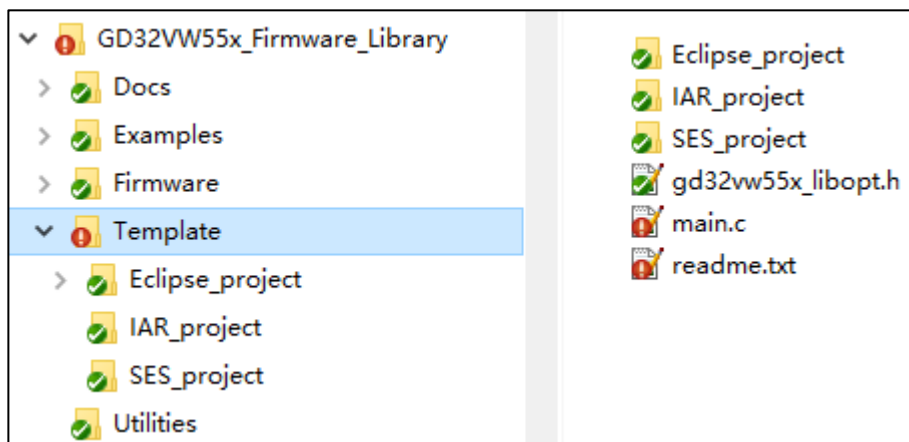


Copy files

Open “Template” folder, keep the folders of “IAR_project”, “Eclipse_project” and “SES_project”, and delete the other files, then copy all the files in

“SPI_master_full duplex_dma” folder to the “Template” subfolder, shown as below:

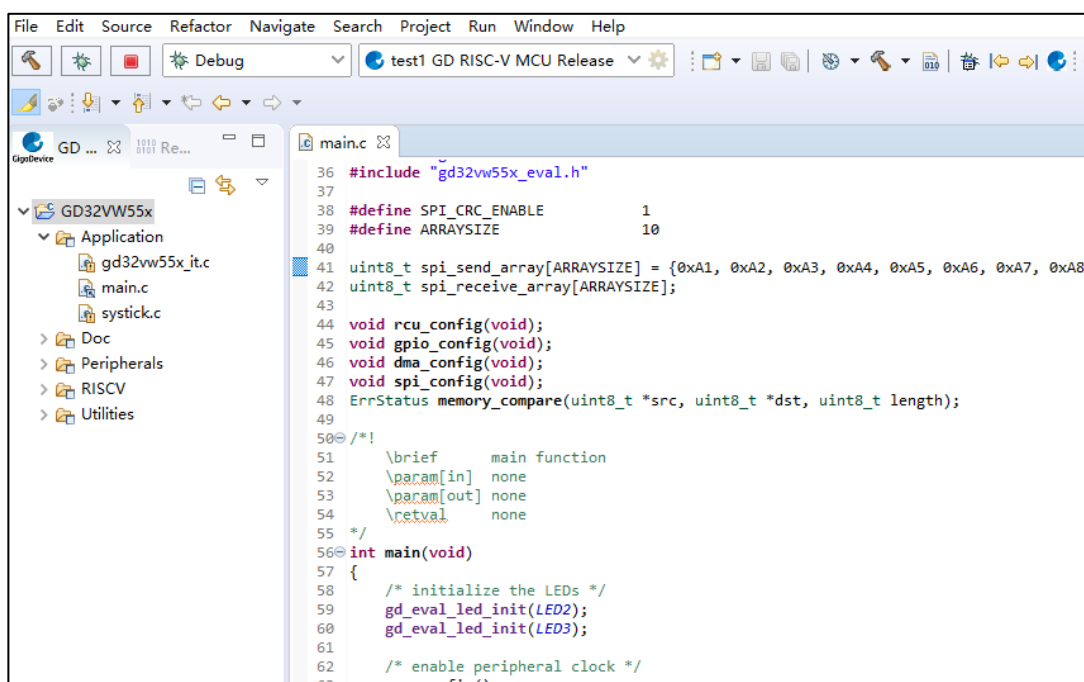
Figure 2-3. Copy the peripheral example files



Open a project

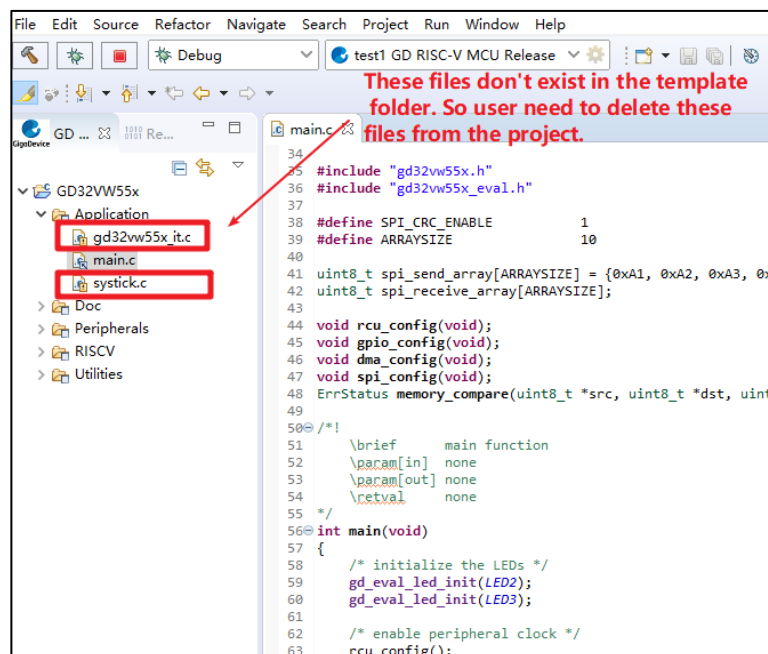
GD provides Eclipse, IAR and SES versions of the project. Depending on the software installed, user can open different projects. For example, open Eclipse and import project in Template folder, shown as below:

Figure 2-4. Open the project file



Because different module and different functions adopt different files, users should add or delete the files in project according to the copied files, shown as below:

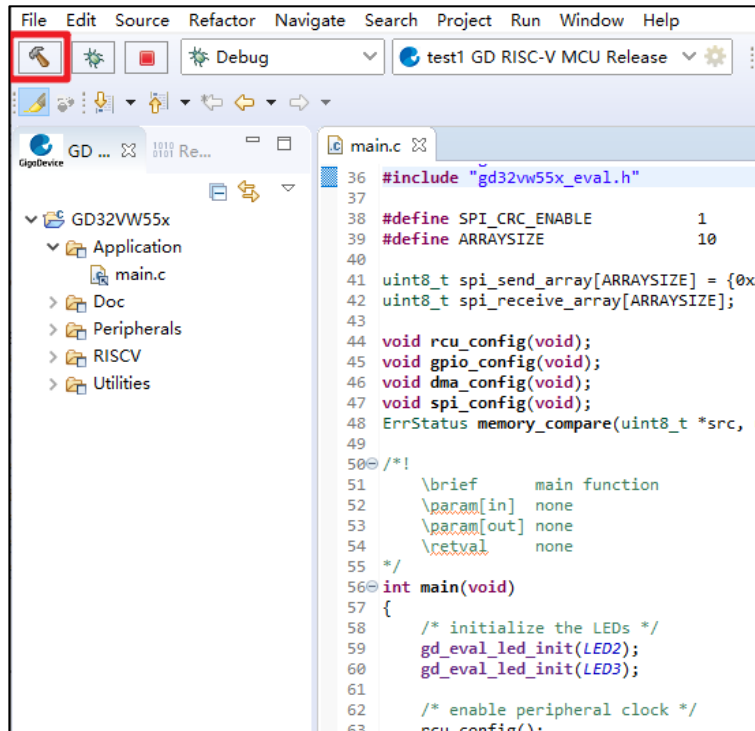
Figure 2-5. Configure project files



Compile-Debug-Download

First compile the project, if there is no error, then select the right jumper cap according to the description of readme, download the project to the target board, and there will be the phenomenon showed accord with the description of readme. Compile operation is shown as below. Debug/download and other memory specific usage of IDE can refer to corresponding Eclipse software user guide.

Figure 2-6. Compile



2.1.5. Utilities Folder

Utilities folder includes files about the firmware examples on evaluation board:

- gd32vw55x_eval.h is the related header file of the evaluation board about running the firmware examples;
- gd32vw55x_eval.c is the related source file of the evaluation board about running the firmware examples.

2.2. File descriptions of Firmware Library

The major files about the firmware library are listed and described in the table below.

Table 2-1. Function descriptions of Firmware Library

Files	Descriptions
gd32vw55x_libopt.h	The header file about all the header files of peripherals. It is the only one file which is necessity to be included in the user's application, to connect the firmware library and the application.
main.c	Example of main function.
gd32vw55x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32vw55x_it.c	Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source

Files	Descriptions
	by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32vw55x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32vw55x_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

3. Firmware Library of Standard Peripherals

3.1. Overview of Firmware Library of Standard Peripherals

The description format of firmware functions are shown as below:

Table 3-1. Peripheral function format of Firmware Library

Function name	Name of peripheral function
Function prototype	Declaration prototype
Function descriptions	Explain the function how to work
Precondition	Requirements should meet before calling this function
The called functions	Other firmware functions called in this function
Input parameter{in}	
Input parameter name	Description
xxxx	Description of input parameters
Output parameter{out}	
Output parameter name	Description
xxxx	Description of output parameters
Return value	
Return value type	The range of return value

3.2. ADC

The 12-bit ADC is an analog-to-digital converter using the successive approximation method. The ADC registers are listed in chapter [3.2.1](#), the ADC firmware functions are introduced in chapter [3.2.2](#).

3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

Table 3-2. ADC Registers

Registers	Descriptions
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx(x=0..3)	Inserted channel data offset register x (x=0..3)
ADC_WDHT	Watchdog high threshold register

Registers	Descriptions
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Routine sequence register 0
ADC_RSQ1	Routine sequence register 1
ADC_RSQ2	Routine sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx (x=0..3)	Inserted data register x(x=0..3)
ADC_RDATA	Routine data register
ADC_OVSAMPCTL	Oversample control register
ADC_CCTL	Common control register

3.2.2. Descriptions of Peripheral functions

ADC firmware functions are listed in the table shown as below:

Table 3-3. ADC firmware function

Function name	Function description
adc_deinit	reset ADC peripheral
adc_clock_config	configure the ADC clock
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_dma_request_after_last_enable	when DMA=1, the DMA engine issues a request at end of each routine conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_tempsensor_vrefint_enable	enable temperature sensor and internal reference voltage channel
adc_tempsensor_vrefint_disable	disable temperature sensor and internal reference voltage channel
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of routine channel group or inserted channel group
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	enable ADC external trigger
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger

Function name	Function description
adc_end_of_conversion_config	configure end of conversion mode
adc_resolution_config	configure ADC resolution
adc_routine_data_read	read ADC routine group data register
adc_inserted_data_read	read ADC routine group data register
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag

adc_deinit

The description of adc_deinit is shown as below:

Table 3-4. Function adc_deinit

Function name	adc_deinit
Function prototype	void adc_deinit(void);
Function descriptions	reset ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset ADC */
adc_deinit();
```

adc_clock_config

The description of adc_clock_config is shown as below:

Table 3-5. Function adc_clock_config

Function name	adc_clock_config
Function prototype	void adc_clock_config(uint32_t prescaler);
Function descriptions	configure the ADC clock
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	configure ADC prescaler ratio
ADC_ADCCCK_PCLK2_DIV2	PCLK2 div2
ADC_ADCCCK_PCLK2_DIV4	PCLK2 div4
ADC_ADCCCK_PCLK2_DIV6	PCLK2 div6
ADC_ADCCCK_PCLK2_DIV8	PCLK2 div8
ADC_ADCCCK_HCLK_DIV5	HCLK div5
ADC_ADCCCK_HCLK_DIV6	HCLK div6
ADC_ADCCCK_HCLK_DIV10	HCLK div10
ADC_ADCCCK_HCLK_DIV20	HCLK div20
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC clock */
adc_clock_config(ADC_ADCCCK_PCLK2_DIV8);
```

adc_enable

The description of adc_enable is shown as below:

Table 3-6. Function adc_enable

Function name	adc_enable
Function prototype	void adc_enable(void);

Function descriptions	enable ADC interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC */
```

```
adc_enable();
```

adc_disable

The description of adc_disable is shown as below:

Table 3-7. Function adc_disable

Function name	adc_disable
Function prototype	void adc_disable(void);
Function descriptions	disable ADC interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC */
```

```
adc_disable();
```

adc_dma_mode_enable

The description of adc_dma_mode_enable is shown as below:

Table 3-8. Function adc_dma_mode_enable

Function name	adc_dma_mode_enable
Function prototype	void adc_dma_mode_enable(void);
Function descriptions	enable ADC DMA request

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC DMA request */
```

```
adc_dma_mode_enable();
```

adc_dma_mode_disable

The description of adc_dma_mode_disable is shown as below:

Table 3-9. Function adc_dma_mode_disable

Function name	adc_dma_mode_disable
Function prototype	void adc_dma_mode_disable(void);
Function descriptions	disable ADC DMA request
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC DMA request */
```

```
adc_dma_mode_disable();
```

adc_dma_request_after_last_enable

The description of adc_dma_request_after_last_enable is shown as below:

Table 3-10. Function adc_dma_request_after_last_enable

Function name	adc_dma_request_after_last_enable
Function prototype	void adc_dma_request_after_last_enable(void);
Function descriptions	when DMA=1, the DMA engine issues a request at end of each routine conversion

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when DMA=1, the DMA engine issues a request at end of each routine conversion */
```

```
adc_dma_request_after_last_enable();
```

adc_dma_request_after_last_disable

The description of adc_dma_request_after_last_disable is shown as below:

Table 3-11. Function adc_dma_request_after_last_disable

Function name	adc_dma_request_after_last_disable
Function prototype	void adc_dma_request_after_last_disable(void);
Function descriptions	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the DMA engine is disabled after the end of transfer signal from DMA is detected */
```

```
adc_dma_request_after_last_enable();
```

adc_discontinuous_mode_config

The description of adc_discontinuous_mode_config is shown as below:

Table 3-12. Function adc_discontinuous_mode_config

Function name	adc_discontinuous_mode_config
Function prototype	void adc_discontinuous_mode_config(uint8_t adc_channel_group, uint8_t length);

Function descriptions	configure ADC discontinuous mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_CHANNEL_DISCON_DISABLE</i>	disable discontinuous mode of routine and inserted channel
Input parameter{in}	
length	number of conversions in discontinuous mode, the number can be 1..9 for routine channel, the number has no effect for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC routine channel group discontinuous mode */
```

```
adc_discontinuous_mode_config(ADC_ROUTINE_CHANNEL, 6);
```

adc_special_function_config

The description of adc_special_function_config is shown as below:

Table 3-13. Function adc_special_function_config

Function name	adc_special_function_config
Function prototype	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
Function descriptions	enable or disable ADC special function
Precondition	-
The called functions	-
Input parameter{in}	
function	the function to config
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
Input parameter{in}	
newvalue	control value

<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC scan mode */
```

```
adc_special_function_config(ADC_SCAN_MODE, ENABLE);
```

adc_tempsensor_vrefint_enable

The description of adc_tempsensor_vrefint_enable is shown as below:

Table 3-14. Function adc_enable

Function name	adc_tempsensor_vrefint_enable
Function prototype	void adc_tempsensor_vrefint_enable (void);
Function descriptions	enable the temperature sensor and Vrefint channel
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the temperature sensor and Vrefint channel */
```

```
adc_tempsensor_vrefint_enable ();
```

adc_tempsensor_vrefint_disable

The description of adc_tempsensor_vrefint_disable is shown as below:

Table 3-15. Function adc_disable

Function name	adc_tempsensor_vrefint_disable
Function prototype	void adc_tempsensor_vrefint_disable (void);
Function descriptions	disable the temperature sensor and Vrefint channel
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the temperature sensor and Vrefint channel */
```

```
adc_tempsensor_vrefint_disable ();
```

adc_data_alignment_config

The description of adc_data_alignment_config is shown as below:

Table 3-16. Function adc_data_alignment_config

Function name	adc_data_alignment_config
Function prototype	void adc_data_alignment_config(uint32_t data_alignment);
Function descriptions	configure ADC data alignment
Precondition	-
The called functions	-
Input parameter{in}	
data_alignment	data alignment select
ADC_DATAALIGN_ RIGHT	right alignment
ADC_DATAALIGN_ LEFT	left alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC data alignment */
```

```
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

adc_channel_length_config

The description of adc_channel_length_config is shown as below:

Table 3-17. Function adc_channel_length_config

Function name	adc_channel_length_config
Function prototype	void adc_channel_length_config(uint8_t adc_channel_group, uint32_t length);

Function descriptions	configure the length of routine channel group or inserted channel group
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_ROUTINE_CHAN NEL</i>	routine channel group
<i>ADC_INSERTED_ CHANNEL</i>	inserted channel group
Input parameter{in}	
length	the length of the channel, routine channel 1-9, inserted channel 1-4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the length of ADC routine channel */
```

```
adc_channel_length_config(ADC_ROUTINE_CHANNEL, 4);
```

adc_routine_channel_config

The description of adc_routine_channel_config is shown as below:

Table 3-18. Function adc_routine_channel_config

Function name	adc_routine_channel_config
Function prototype	void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC routine channel
Precondition	-
The called functions	-
Input parameter{in}	
rank	the routine group sequence rank, this parameter must be between 0 to 8
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x (x=0..10)</i>	ADC Channelx (x=0..10)
Input parameter{in}	
sample_time	the sample time value
<i>ADC_SAMPLETIME_1 POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_2 POINT5</i>	2.5 cycles

ADC_SAMPLETIME_1 4POINT5	14.5 cycles
ADC_SAMPLETIME_2 7POINT5	27.5 cycles
ADC_SAMPLETIME_5 5POINT5	55.5 cycles
ADC_SAMPLETIME_8 3POINT5	83.5 cycles
ADC_SAMPLETIME_1 11POINT5	111.5 cycles
ADC_SAMPLETIME_1 43POINT5	143.5 cycles
ADC_SAMPLETIME_4 79POINT5	479.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC routine channel */
```

```
adc_routine_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_14POINT5);
```

adc_inserted_channel_config

The description of adc_inserted_channel_config is shown as below:

Table 3-19. Function adc_inserted_channel_config

Function name	adc_inserted_channel_config
Function prototype	void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC inserted channel
Precondition	-
The called functions	-
Input parameter{in}	
rank	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x (x=0..10)	ADC Channelx (x=0..10)
Input parameter{in}	
sample_time	the sample time value
ADC_SAMPLETIME_1	1.5 cycles

<i>POINT5</i>	
<i>ADC_SAMPLETIME_2</i> <i>POINT5</i>	2.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>4POINT5</i>	14.5 cycles
<i>ADC_SAMPLETIME_2</i> <i>7POINT5</i>	27.5 cycles
<i>ADC_SAMPLETIME_5</i> <i>5POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_8</i> <i>3POINT5</i>	83.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>11POINT5</i>	111.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>43POINT5</i>	143.5 cycles
<i>ADC_SAMPLETIME_4</i> <i>79POINT5</i>	479.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC inserted channel */
```

```
adc_inserted_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_14POINT5);
```

adc_inserted_channel_offset_config

The description of `adc_inserted_channel_offset_config` is shown as below:

Table 3-20. Function `adc_inserted_channel_offset_config`

Function name	<code>adc_inserted_channel_offset_config</code>
Function prototype	<code>void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);</code>
Function descriptions	configure ADC inserted channel offset
Precondition	-
The called functions	-
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be between 0 to 4095

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC inserted channel offset */
```

```
adc_inserted_channel_offset_config(ADC_INSERTED_CHANNEL_0, 100);
```

adc_external_trigger_config

The description of adc_external_trigger_config is shown as below:

Table 3-21. Function adc_external_trigger_config

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint8_t channel_group, uint32_t trigger_mode);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
trigger_mode	external trigger mode
EXTERNAL_TRIGGER_DISABLE	external trigger disable
EXTERNAL_TRIGGER_RISING	rising edge of external trigger
EXTERNAL_TRIGGER_FALLING	falling edge of external trigger
EXTERNAL_TRIGGER_RISING_FALLING	rising and falling edge of external trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC inserted channel group external trigger */
```

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL,
EXTERNAL_TRIGGER_RISING);
```

adc_external_trigger_source_config

The description of adc_external_trigger_source_config is shown as below:

Table 3-22. Function adc_external_trigger_source_config

Function name	adc_external_trigger_source_config
Function prototype	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
Function descriptions	configure ADC external trigger source
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
external_trigger_source	routine or inserted group trigger source
ADC_EXTTRIG_ROUTINE_T0_CH0	TIMER0 CH0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T0_CH1	TIMER0 CH1 event select for routine channel
ADC_EXTTRIG_ROUTINE_T0_CH2	TIMER0 CH2 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH1	TIMER1 CH1 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH2	TIMER1 CH2 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH3	TIMER1 CH3 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_TRGO	TIMER1 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_T2_CH0	TIMER2 CH0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T2_TRGO	TIMER2 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_EXTI_11	extiline 11 event select for routine channel
ADC_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3 event select for inserted channel

<i>INSERTED_T0_CH3</i>	
<i>ADC_EXTTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T2_CH1</i>	TIMER2 CH1 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T2_CH3</i>	TIMER3 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC routine channel external trigger source */
```

```
adc_external_trigger_source_config (ADC_ROUTINE_CHANNEL,  
ADC_EXTTRIG_ROUTINE_T0_CH0);
```

adc_software_trigger_enable

The description of `adc_software_trigger_enable` is shown as below:

Table 3-23. Function `adc_software_trigger_enable`

Function name	<code>adc_software_trigger_enable</code>
Function prototype	<code>void adc_software_trigger_enable(uint8_t adc_channel_group);</code>
Function descriptions	enable ADC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC routine channel group software trigger */
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

adc_end_of_conversion_config

The description of adc_end_of_conversion_config is shown as below:

Table 3-24. Function adc_end_of_conversion_config

Function name	adc_end_of_conversion_config
Function prototype	void adc_end_of_conversion_config(uint8_t end_selection);
Function descriptions	configure end of conversion mode
Precondition	-
The called functions	-
Input parameter{in}	
end_selection	end of conversion mode
ADC_EOC_SET_SEQUENCE	only at the end of a sequence of routine conversions, the EOC bit is set. overflow detection is disabled unless DMA=1.
ADC_EOC_SET_CONVERSION	at the end of each routine conversion, the EOC bit is set. Overflow is detected automatically.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure end of conversion mode */
adc_end_of_conversion_config(ADC_EOC_SET_SEQUENCE);
```

adc_resolution_config

The description of adc_resolution_config is shown as below:

Table 3-25. adc_resolution_config

Function name	adc_resolution_config
Function prototype	void adc_resolution_config(uint32_t resolution);
Function descriptions	configure ADC resolution
Precondition	-
The called functions	-
Input parameter{in}	
resolution	configure ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution

ADC_RESOLUTION_1 0B	10-bit ADC resolution
ADC_RESOLUTION_8 B	8-bit ADC resolution
ADC_RESOLUTION_6 B	6-bit ADC resolution
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC resolution*/
adc_resolution_config(ADC_RESOLUTION_12B);
```

adc_routine_data_read

The description of adc_routine_data_read is shown as below:

Table 3-26. Function adc_routine_data_read

Function name	adc_routine_data_read
Function prototype	uint16_t adc_routine_data_read(void);
Function descriptions	read ADC routine group data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0~0xFFFF)

Example:

```
/* read ADC routine group data register */
uint16_t adc_value = 0;
adc_value = adc_routine_data_read();
```

adc_inserted_data_read

The description of adc_inserted_data_read is shown as below:

Table 3-27. Function adc_inserted_data_read

Function name	adc_inserted_data_read
---------------	------------------------

Function prototype	uint16_t adc_inserted_data_read(uint8_t inserted_channel);
Function descriptions	read ADC inserted group data register
Precondition	-
The called functions	-
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0~0xFFFF)

Example:

```
/* read ADC inserted group data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_inserted_data_read (ADC_INSERTED_CHANNEL_0);
```

adc_watchdog_single_channel_enable

The description of adc_watchdog_single_channel_enable is shown as below:

Table 3-28. Function adc_watchdog_single_channel_enable

Function name	adc_watchdog_single_channel_enable
Function prototype	void adc_watchdog_single_channel_enable(uint8_t adc_channel);
Function descriptions	configure ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x</i> (x=0..10)	ADC channelx(x=0..10)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog single channel */
```

```
adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

adc_watchdog_group_channel_enable

The description of adc_watchdog_group_channel_enable is shown as below:

Table 3-29. Function adc_watchdog_group_channel_enable

Function name	adc_watchdog_group_channel_enable
Function prototype	void adc_watchdog_group_channel_enable(uint8_t adc_channel_group);
Function descriptions	configure ADC analog watchdog group channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_channel_group	the channel group use analog watchdog
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_ROUTINE_INSERTED_CHANNEL	both routine and inserted group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC_ROUTINE_CHANNEL);
```

adc_watchdog_disable

The description of adc_watchdog_disable is shown as below:

Table 3-30. Function adc_watchdog_disable

Function name	adc_watchdog_disable
Function prototype	void adc_watchdog_disable(void);
Function descriptions	disable ADC analog watchdog
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC analog watchdog */
```

```
adc_watchdog_disable();
```

adc_watchdog_threshold_config

The description of adc_watchdog_threshold_config is shown as below:

Table 3-31. Function adc_watchdog_threshold_config

Function name	adc_watchdog_threshold_config
Function prototype	void adc_watchdog_threshold_config(uint16_t low_threshold , uint16_t high_threshold);
Function descriptions	configure ADC analog watchdog threshold
Precondition	-
The called functions	-
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0..4095
Input parameter{in}	
high_threshold	analog watchdog high threshold, 0..4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog threshold */
```

```
adc_watchdog_threshold_config( 0x0400, 0x0A00);
```

adc_oversample_mode_config

The description of adc_oversample_mode_config is shown as below:

Table 3-32. Function adc_oversample_mode_config

Function name	adc_oversample_mode_config
Function prototype	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
Function descriptions	configure ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	ADC oversampling mode
ADC_OVERSAMPLING _ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger

ADC_OVERSAMPLING _ONE_CONVERT	each oversampled conversion for a channel needs a trigger
Input parameter{in}	
shift	ADC oversampling shift
ADC_OVERSAMPLING _SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING _SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_5B	5-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_8B	8-bit oversampling shift
Input parameter{in}	
ratio	ADC oversampling ratio
ADC_OVERSAMPLING _RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING _RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING _RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING _RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING _RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING _RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING _RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING _RATIO_MUL256	oversampling ratio multiple 256
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure ADC oversample mode: 16 times sample, 4 bits shift */
```

```
adc_oversample_mode_config(ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

adc_oversample_mode_enable

The description of adc_oversample_mode_enable is shown as below:

Table 3-33. Function adc_oversample_mode_enable

Function name	adc_oversample_mode_enable
Function prototype	void adc_oversample_mode_enable(void);
Function descriptions	enable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC oversample mode */
```

```
adc_oversample_mode_enable();
```

adc_oversample_mode_disable

The description of adc_oversample_mode_disable is shown as below:

Table 3-34. Function adc_oversample_mode_disable

Function name	adc_oversample_mode_disable
Function prototype	void adc_oversample_mode_disable(void);
Function descriptions	disable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable ADC oversample mode */
```

```
adc_oversample_mode_disable ();
```

adc_flag_get

The description of adc_flag_get is shown as below:

Table 3-35. Function adc_flag_get

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t flag);
Function descriptions	get the ADC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	the adc flag bits
ADC_FLAG_WDE	analog watchdog event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of routine channel group
ADC_FLAG_ROVF	routine data register overflow flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC analog watchdog flag bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC_FLAG_WDE);
```

adc_flag_clear

The description of adc_flag_clear is shown as below:

Table 3-36. Function adc_flag_clear

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t flag);
Function descriptions	clear the ADC flag bits
Precondition	-

The called functions	-
Input parameter{in}	
flag	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of routine channel group
<i>ADC_FLAG_ROVF</i>	routine data register overflow flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC analog watchdog flag bits */
```

```
adc_flag_clear(ADC_FLAG_WDE);
```

adc_interrupt_enable

The description of adc_interrupt_enable is shown as below:

Table 3-37. Function adc_interrupt_enable

Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the adc interrupt
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_ROVF</i>	routine data register overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC analog watchdog interrupt */
```

```
adc_interrupt_enable(ADC_INT_WDE);
```

adc_interrupt_disable

The description of adc_interrupt_disable is shown as below:

Table 3-38. Function adc_interrupt_disable

Function name	adc_interrupt_disable
Function prototype	void adc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the adc interrupt
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
ADC_INT_ROVF	routine data register overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC analog watchdog interrupt */
```

```
adc_interrupt_disable(ADC_INT_WDE);
```

adc_interrupt_flag_get

The description of adc_interrupt_flag_get is shown as below:

Table 3-39. Function adc_interrupt_flag_get

Function name	adc_interrupt_flag_get
Function prototype	FlagStatus adc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	the adc interrupt bits
ADC_INT_FLAG_WDE	analog watchdog interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
ADC_INT_FLAG_ROVF	routine data register overflow interrupt
Output parameter{out}	

-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC analog watchdog interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

adc_interrupt_flag_clear

The description of adc_interrupt_flag_clear is shown as below:

Table 3-40. Function adc_interrupt_flag_clear

Function name	adc_interrupt_flag_clear
Function prototype	void adc_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	the adc interrupt bits
ADC_INT_FLAG_WDE	analog watchdog interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
ADC_INT_FLAG_ROVF	routine data register overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC analog watchdog interrupt bits*/
```

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

3.3. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter [3.3.1](#) the CAU firmware functions are introduced in chapter [3.3.2](#).

3.3.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

Table 3-41. CAU Registers

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register
CAU_GCMCCMCT XSx (x=0..7)	GCM or CCM mode context switch register x
CAU_GCMCTXSx (x=0..7)	GCM mode context switch register x

3.3.2. Descriptions of Peripheral functions

CAU firmware functions are listed in the table shown as below:

Table 3-42. CAU firmware function

Function name	Function description
cau_deinit	reset the CAU peripheral
cau_struct_para_init	initialize the CAU encrypt and decrypt parameter struct with the default values
cau_key_struct_para_init	initialize the key parameter struct with the default values
cau_iv_struct_para_init	initialize the vectors parameter struct with the default values

Function name	Function description
cau_context_struct_para_init	initialize the context parameter struct with the default values
cau_enable	enable the CAU peripheral
cau_disable	disable the CAU peripheral
cau_dma_enable	enable the CAU DMA interface
cau_dma_disable	disable the CAU DMA interface
cau_init	initialize the CAU peripheral
cau_aes_keysize_config	configure key size if use AES algorithm
cau_key_init	initialize the key parameters
cau_iv_init	initialize the vectors parameters
cau_phase_config	configure phase
cau_fifo_flush	flush the IN and OUT FIFOs
cau_enable_state_get	return whether CAU peripheral is enabled or disabled
cau_data_write	write data to the IN FIFO
cau_data_read	return the last data entered into the output FIFO
cau_context_save	save context before context switching
cau_context_restore	restore context after context switching
cau_aes_ecb	encrypt and decrypt using AES in ECB mode
cau_aes_cbc	encrypt and decrypt using AES in CBC mode
cau_aes_ctr	encrypt and decrypt using AES in CTR mode
cau_aes_cfb	encrypt and decrypt using AES in CFB mode
cau_aes_ofb	encrypt and decrypt using AES in OFB mode
cau_aes_gcm	encrypt and decrypt using AES in GCM mode
cau_aes_ccm	encrypt and decrypt using AES in CCM mode
cau_tdes_ecb	encrypt and decrypt using TDES in ECB mode
cau_tdes_cbc	encrypt and decrypt using TDES in CBC mode
cau_des_ecb	encrypt and decrypt using DES in ECB mode
cau_des_cbc	encrypt and decrypt using DES in CBC mode
cau_flag_get	get the CAU flag status
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag

Structure cau_key_parameter_struct

Table 3-43. Structure cau_key_parameter_struct

Member name	Function description
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low

key_3_high	key 3 high
key_3_low	key 3 low

Structure cau_iv_parameter_struct

Table 3-44. Structure cau_iv_parameter_struct

Member name	Function description
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low

Structure cau_context_parameter_struct

Table 3-45. Structure cau_context_parameter_struct

Member name	Function description
ctl_config	current configuration
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low
gcmccmctxs[8]	GCM or CCM mode context switch
gcmctxs[8]	GCM mode context switch

Structure cau_parameter_struct

Table 3-46. Structure cau_parameter_struct

Member name	Function description
alg_dir	algorithm directory
*key	key
key_size	key size in bytes
*iv	initialization vector
iv_size	iv size in bytes
*input	input data
in_length	input data length in bytes
*aad	additional authentication data

aad_size	aad size
----------	----------

cau_deinit

The description of cau_deinit is shown as below:

Table 3-47. Function cau_deinit

Function name	cau_deinit
Function prototype	void cau_deinit(void);
Function descriptions	reset the CAU peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the CAU peripheral */
```

```
cau_deinit( );
```

cau_struct_para_init

The description of cau_struct_para_init is shown as below:

Table 3-48. Function cau_struct_para_init

Function name	cau_struct_para_init
Function prototype	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
Function descriptions	initialize the CAU encrypt and decrypt parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Return value	
-	-

Example:

```
cau_parameter_struct text;
```



```
/* initialize CAU encrypt and decrypt parameter struct */
```

```
cau_struct_para_init(&text);
```

cau_key_struct_para_init

The description of cau_key_struct_para_init is shown as below:

Table 3-49. Function cau_key_struct_para_init

Function name	cau_key_struct_para_init
Function prototype	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
Function descriptions	initialize the key parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
key_initpara	the key parameters, refer to structure Table 3-43. Structure cau_key_parameter_struct
Return value	
-	-

Example:

```
/* initialize the key parameter struct */
```

```
cau_key_parameter_struct key_initpara;
```

```
cau_key_struct_para_init(&key_initpara);
```

cau_iv_struct_para_init

The description of cau_iv_struct_para_init is shown as below:

Table 3-50. Function cau_iv_struct_para_init

Function name	cau_iv_struct_para_init
Function prototype	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara);
Function descriptions	initialize the vectors parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
iv_initpara	the vectors parameter, refer to structure Table 3-44. Structure cau_iv_parameter_struct
Return value	
-	-

Example:

```
/* initialize the vectors parameter struct */

cau_iv_parameter_struct iv_initpara;

cau_iv_struct_para_init(&iv_initpara);
```

cau_context_struct_para_init

The description of cau_context_struct_para_init is shown as below:

Table 3-51. Function cau_context_struct_para_init

Function name	cau_context_struct_para_init
Function prototype	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context);
Function descriptions	initialize the context parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
cau_context	the context parameter, refer to structure Table 3-45. Structure cau context parameter struct
Return value	
-	-

Example:

```
/* initialize the context parameter struct */

cau_context_parameter_struct context_initpara;

cau_context_struct_para_init (&context_initpara);
```

cau_enable

The description of cau_enable is shown as below:

Table 3-52. Function cau_enable

Function name	cau_enable
Function prototype	void cau_enable(void);
Function descriptions	enable the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable the CAU peripheral */
```

```
cau_enable();
```

cau_disable

The description of cau_disable is shown as below:

Table 3-53. Function cau_disable

Function name	cau_disable
Function prototype	void cau_disable(void);
Function descriptions	disable the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CAU peripheral */
```

```
cau_disable();
```

cau_dma_enable

The description of cau_dma_enable is shown as below:

Table 3-54. Function cau_dma_enable

Function name	cau_dma_enable
Function prototype	void cau_dma_enable(uint32_t dma_req);
Function descriptions	enable the CAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be enabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CAU DMA interface */
cau_dma_enable(CAU_DMA_INFIFO);
```

cau_dma_disable

The description of cau_dma_disable is shown as below:

Table 3-55. Function cau_dma_disable

Function name	cau_dma_disable
Function prototype	void cau_dma_disable(uint32_t dma_req);
Function descriptions	disable the CAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be disabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CAU DMA interface */
cau_dma_disable(CAU_DMA_INFIFO);
```

cau_init

The description of cau_init is shown as below:

Table 3-56. Function cau_init

Function name	cau_init
Function prototype	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping);
Function descriptions	initialize the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	

alg_dir	algorithm direction
<i>CAU_ENCRYPT</i>	encrypt
<i>CAU_DECRYPT</i>	decrypt
Input parameter{in}	
algo_mode	algorithm mode selection
<i>CAU_MODE_TDES_ECB</i>	TDES-ECB (3DES Electronic codebook)
<i>CAU_MODE_TDES_CBC</i>	TDES-CBC (3DES Cipher block chaining)
<i>CAU_MODE_DES_EC</i> <i>B</i>	DES-ECB (simple DES Electronic codebook)
<i>CAU_MODE_DES_CB</i> <i>C</i>	DES-CBC (simple DES Cipher block chaining)
<i>CAU_MODE_AES_EC</i> <i>B</i>	AES-ECB (AES Electronic codebook)
<i>CAU_MODE_AES_CB</i> <i>C</i>	AES-CBC (AES Cipher block chaining)
<i>CAU_MODE_AES_CT</i> <i>R</i>	AES-CTR (AES counter mode)
<i>CAU_MODE_AES_KE</i> <i>Y</i>	AES decryption key preparation mode
<i>CAU_MODE_AES_GC</i> <i>M</i>	AES-GCM (AES Galois/counter mode)
<i>CAU_MODE_AES_CC</i> <i>M</i>	AES-CCM (AES combined cipher machine mode)
<i>CAU_MODE_AES_CF</i> <i>B</i>	AES-CFB (cipher feedback mode)
<i>CAU_MODE_AES_OF</i> <i>B</i>	AES-OFB (output feedback mode)
Input parameter{in}	
swapping	data swapping selection
<i>CAU_SWAPPING_32BIT</i>	no swapping
<i>CAU_SWAPPING_16BIT</i> <i>T</i>	half-word swapping
<i>CAU_SWAPPING_8BIT</i>	bytes swapping
<i>CAU_SWAPPING_1BIT</i>	bit swapping
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the CAU peripheral */
```

```
cau_init(CAU_ENCRYPT, CAU_MODE_TDES_ECB, CAU_SWAPPING_32BIT);
```

cau_aes_keysize_config

The description of cau_aes_keysize_config is shown as below:

Table 3-57. Function cau_aes_keysize_config

Function name	cau_aes_keysize_config
Function prototype	void cau_aes_keysize_config(uint32_t key_size);
Function descriptions	configure key size if use AES algorithm
Precondition	-
The called functions	-
Input parameter{in}	
key_size	key length selection when aes mode
CAU_KEYSIZE_128BIT	128 bit key length
CAU_KEYSIZE_192BIT	192 bit key length
CAU_KEYSIZE_256BIT	256 bit key length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure key size if used AES algorithm */
```

```
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

cau_key_init

The description of cau_key_init is shown as below:

Table 3-58. Function cau_key_init

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure Table 3-43. Structure cau_key_parameter_struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* initialize the key parameters */

cau_key_parameter_struct key_initpara;

cau_key_init (&key_initpara);
```

cau_iv_init

The description of cau_iv_init is shown as below:

Table 3-59. Function cau_iv_init

Function name	cau_iv_init
Function prototype	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
Function descriptions	initialize the vectors parameters
Precondition	-
The called functions	-
Input parameter{in}	
iv_initpara	the vectors parameters, refer to structure Table 3-44. Structure cau_iv_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the vectors parameters */

cau_iv_parameter_struct iv_initpara;

cau_iv_init(&iv_initpara);
```

cau_phase_config

The description of cau_phase_config is shown as below:

Table 3-60. Function cau_phase_config

Function name	cau_phase_config
Function prototype	void cau_phase_config(uint32_t phase);
Function descriptions	configure phase
Precondition	-
The called functions	-
Input parameter{in}	
phase	gcm or ccm phase
CAU_PREPARE_PHAS	prepare phase

<i>E</i>	
<i>CAU_AAD_PHASE</i>	AAD phase
<i>CAU_ENCRYPT_DEC RYPT_PHASE</i>	encryption/decryption phase
<i>CAU_TAG_PHASE</i>	tag phase
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select prepare phase */
cau_phase_config(CAU_PREPARE_PHASE);
```

cau_fifo_flush

The description of cau_fifo_flush is shown as below:

Table 3-61. Function cau_fifo_flush

Function name	cau_fifo_flush
Function prototype	void cau_fifo_flush(void);
Function descriptions	flush the IN and OUT FIFOs
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* flush the IN and OUT FIFOs */
cau_fifo_flush( );
```

cau_enable_state_get

The description of cau_enable_state_get is shown as below:

Table 3-62. Function cau_enable_state_get

Function name	cau_enable_state_get
Function prototype	ControlStatus cau_enable_state_get(void);
Function descriptions	return whether CAU peripheral is enabled or disabled

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ControlStatus	ENABLE or DISABLE

Example:

```
/* return whether CAU peripheral is enabled or disabled */
```

```
ControlStatus state = cau_enable_state_get();
```

cau_data_write

The description of cau_data_write is shown as below:

Table 3-63. Function cau_data_write

Function name	cau_data_write
Function prototype	void cau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write: 0 - 0xFFFFFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data to the IN FIFO */
```

```
cau_data_write(0x10);
```

cau_data_read

The description of cau_data_read is shown as below:

Table 3-64. Function cau_data_read

Function name	cau_data_read
Function prototype	uint32_t cau_data_read(void);
Function descriptions	return the last data entered into the output FIFO
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the last data entered into the output FIFO */
```

```
uint32_t data;
```

```
data = cau_data_read();
```

cau_context_save

The description of cau_context_save is shown as below:

Table 3-65. Function cau_context_save

Function name	cau_context_save
Function prototype	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara)
Function descriptions	save context before context switching
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure Table 3-43. Structure cau_key_parameter_struct
Output parameter{out}	
cau_context	the context, refer to structure Table 3-45. Structure cau_context_parameter_struct
Return value	
-	-

Example:

```
cau_context_parameter_struct context;
```

```
cau_key_parameter_struct key;
```

```
cau_parameter_struct cau_parameter;
```

```
uint32_t keyaddr;
```

```
.....
```

```
keyaddr = (uint32_t)(cau_parameter->key);
```

```

cau_key_struct_para_init(&key);

key.key_1_high = __REV(*(uint32_t*)(keyaddr));

keyaddr += 4U;

key.key_1_low = __REV(*(uint32_t*)(keyaddr));

/* save context before context switching */

cau_context_save(&context, &key);

```

cau_context_restore

The description of cau_context_restore is shown as below:

Table 3-66. Function cau_context_restore

Function name	cau_context_restore
Function prototype	void cau_context_restore(cau_context_parameter_struct *cau_context);
Function descriptions	restore context after context switching
Precondition	-
The called functions	-
Input parameter{in}	
cau_context	the context, refer to structure Table 3-45. Structure cau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

cau_context_parameter_struct context;

.....

cau_context_save(&context, &key);

.....

/* restore context after context switching */

cau_context_restore (&context);

```

cau_aes_ecb

The description of cau_aes_ecb is shown as below:

Table 3-67. Function cau_aes_ecb

Function name	cau_aes_ecb
Function prototype	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t

	*output);
Function descriptions	encrypt and decrypt using AES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);
```

cau_aes_cbc

The description of cau_aes_cbc is shown as below:

Table 3-68. Function cau_aes_cbc

Function name	cau_aes_cbc
Function prototype	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using AES in CBC mode

Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CBC mode */

status = cau_aes_cbc(&text, encrypt_result);
```

cau_aes_ctr

The description of cau_aes_ctr is shown as below:

Table 3-69. Function cau_aes_ctr

Function name	cau_aes_ctr
Function prototype	ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using AES in CTR mode
Precondition	-

The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CTR mode */

status = cau_aes_ctr(&text, encrypt_result);
```

cau_aes_cfb

The description of cau_aes_cfb is shown as below:

Table 3-70. Function cau_aes_cfb

Function name	cau_aes_cfb
Function prototype	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using AES in CFB mode
Precondition	-
The called functions	-

Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_cfb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_cfb_parameter);

/* encryption in CFB mode */

cau_cfb_parameter.alg_dir   = CAU_ENCRYPT;
cau_cfb_parameter.key       = (uint8_t *)key_128;
cau_cfb_parameter.key_size  = KEY_SIZE;
cau_cfb_parameter.iv        = (uint8_t *)vectors;
cau_cfb_parameter.iv_size   = IV_SIZE;
cau_cfb_parameter.input     = (uint8_t *)plaintext;
cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);
```

cau_aes_ofb

The description of cau_aes_ofb is shown as below:

Table 3-71. Function cau_aes_ofb

Function name	cau_aes_ofb
Function prototype	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using AES in OFB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46.

	<u>Structure cau_parameter_struct</u>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_ofb_parameter);

/* encryption in OFB mode */

cau_ofb_parameter.alg_dir    = CAU_ENCRYPT;
cau_ofb_parameter.key        = (uint8_t *)key_128;
cau_ofb_parameter.key_size   = KEY_SIZE;
cau_ofb_parameter.iv         = (uint8_t *)vectors;
cau_ofb_parameter.iv_size    = IV_SIZE;
cau_ofb_parameter.input      = (uint8_t *)plaintext;
cau_ofb_parameter.in_length  = PLAINTEXT_SIZE;

status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);
```

cau_aes_gcm

The description of cau_aes_gcm is shown as below:

Table 3-72. Function cau_aes_gcm

Function name	cau_aes_gcm
Function prototype	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag)
Function descriptions	encrypt and decrypt using AES in GCM mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <u>Table 3-46. Structure cau_parameter_struct</u>
Output parameter{out}	

output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_gcm_parameter);

/* encryption in GCM mode */

cau_gcm_parameter.alg_dir    = CAU_ENCRYPT;

cau_gcm_parameter.key        = (uint8_t *)key_128;

cau_gcm_parameter.key_size   = KEY_SIZE;

cau_gcm_parameter.iv         = (uint8_t *)vectors;

cau_gcm_parameter.iv_size    = IV_SIZE;

cau_gcm_parameter.input      = (uint8_t *)plaintext;

cau_gcm_parameter.in_length  = PLAINTEXT_SIZE;

cau_gcm_parameter.aad        = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);

```

cau_aes_ccm

The description of cau_aes_ccm is shown as below:

Table 3-73. Function cau_aes_ccm

Function name	cau_aes_ccm
Function prototype	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[])
Function descriptions	encrypt and decrypt using AES in CCM mode
Precondition	-
The called functions	-

Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Input parameter{in}	
tag_size	tag size (in bytes)
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Output parameter{out}	
aad_buf	pointer to the user buffer used when formatting aad block
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct cau_ccm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t ccm_tag[CCM_TAG_SIZE];

uint8_t aad_buf[AAD_SIZE + 21];

ErrStatus status;

.....

cau_struct_para_init(&cau_ccm_parameter);

/* encryption in CCM mode */

cau_ccm_parameter.alg_dir    = CAU_ENCRYPT;
cau_ccm_parameter.key        = (uint8_t *)ccm_key_128;
cau_ccm_parameter.key_size   = KEY_SIZE;
cau_ccm_parameter.iv         = (uint8_t *)ccm_vectors;
cau_ccm_parameter.iv_size    = CCM_IV_SIZE;
cau_ccm_parameter.input      = (uint8_t *)plaintext;
cau_ccm_parameter.in_length  = PLAINTEXT_SIZE;
cau_ccm_parameter.aad        = (uint8_t *)aadmessage;
cau_ccm_parameter.aad_size   = AAD_SIZE;

status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE,
(uint8_t *)aad_buf);

```

cau_tdes_ecb

The description of cau_tdes_ecb is shown as below:

Table 3-74. Function cau_tdes_ecb

Function name	cau_tdes_ecb
Function prototype	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using TDES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir = CAU_ENCRYPT;

text.key = tdes_key;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_tdes_ecb(&text, encrypt_result);
```

cau_tdes_cbc

The description of cau_tdes_cbc is shown as below:

Table 3-75. Function cau_tdes_cbc

Function name	cau_tdes_cbc
Function prototype	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)

Function descriptions	encrypt and decrypt using TDES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir = CAU_ENCRYPT;

text.key = tdes_key;

text.iv = vectors;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_tdes_cbc(&text, encrypt_result);
```

cau_des_ecb

The description of cau_des_ecb is shown as below:

Table 3-76. Function cau_des_ecb

Function name	cau_des_ecb
Function prototype	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using DES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct

	<u>Structure cau_parameter_struct</u>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir    = CAU_ENCRYPT;

text.key        = des_key;

text.input      = plaintext;

text.in_length  = DATA_SIZE;

/* encryption in ECB mode */

status = cau_des_ecb(&text, encrypt_result);
```

cau_des_cbc

The description of cau_des_cbc is shown as below:

Table 3-77. Function cau_des_cbc

Function name	cau_des_cbc
Function prototype	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
Function descriptions	encrypt and decrypt using DES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <u>Table 3-46. Structure cau_parameter_struct</u>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir    = CAU_ENCRYPT;

text.key        = des_key;

text.iv         = vectors;

text.input      = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_des_cbc(&text, encrypt_result);

```

cau_flag_get

The description of cau_flag_get is shown as below:

Table 3-78. Function cau_flag_get

Function name	cau_flag_get
Function prototype	FlagStatus cau_flag_get(uint32_t flag);
Function descriptions	get the CAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	CAU flag status
CAU_FLAG_INFIFO_EMPTY	input FIFO empty
CAU_FLAG_INFIFO_NO_FULL:	input FIFO is not full
CAU_FLAG_OUTFIFO_NO_EMPTY	output FIFO not empty
CAU_FLAG_OUTFIFO_FULL	output FIFO is full
CAU_FLAG_BUSY	the CAU core is busy
CAU_FLAG_INFIFO	input FIFO flag status
CAU_FLAG_OUTFIFO	output FIFO flag status
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* get the CAU flag status */
```

```
FlagStatus status;
```

```
status = cau_flag_get (CAU_FLAG_INFIFO_EMPTY);
```

cau_interrupt_enable

The description of cau_interrupt_enable is shown as below:

Table 3-79. Function cau_interrupt_enable

Function name	cau_interrupt_enable
Function prototype	void cau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be enabled
CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable cau interrupt */
```

```
cau_interrupt_enable (CAU_INT_INFIFO);
```

cau_interrupt_disable

The description of cau_interrupt_disable is shown as below:

Table 3-80. Function cau_interrupt_disable

Function name	cau_interrupt_disable
Function prototype	void cau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be disabled

CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable cau interrupt */
```

```
cau_interrupt_disable(CAU_INT_INFIFO);
```

cau_interrupt_flag_get

The description of cau_interrupt_flag_get is shown as below:

Table 3-81. Function cau_interrupt_flag_get

Function name	cau_interrupt_flag_get
Function prototype	FlagStatus cau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	CAU interrupt flag
CAU_INT_FLAG_INFIFO	input FIFO interrupt
CAU_INT_FLAG_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the CAU interrupt flag status */
```

```
FlagStatus status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

3.4. CRC

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. The CRC registers are listed in chapter [3.4.1](#), the CRC firmware functions are introduced in chapter [3.4.2](#).

3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

Table 3-82. CRC Registers

Registers	Descriptions
CRC_DATA	CRC data register
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register

3.4.2. Descriptions of Peripheral functions

CRC firmware functions are listed in the table shown as below:

Table 3-83. CRC firmware function

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_data_register_reset	reset data register to the value of initializaiton data register
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_single_data_calculate	CRC calculate a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

crc_deinit

The description of crc_deinit is shown as below:

Table 3-84. Function crc_deinit

Function name	crc_deinit
Function prototype	void crc_deinit(void);
Function descriptions	deinit CRC calculation unit
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset crc */
crc_deinit();
```

crc_data_register_reset

The description of crc_data_register_reset is shown as below:

Table 3-85. Function crc_data_register_reset

Function name	crc_data_register_reset
Function prototype	void crc_data_register_reset(void);
Function descriptions	reset data register(CRC_DATA) to the value of 0xFFFFFFFF
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset crc data register */
crc_data_register_reset ();
```

crc_data_register_read

The description of crc_data_register_read is shown as below:

Table 3-86. Function crc_data_register_read

Function name	crc_data_register_read
Function prototype	uint32_t crc_data_register_read(void);
Function descriptions	read the value of the data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value of the data register (0-0xFFFFFFFF)

Example:

```
/* read crc data register */
uint32_t crc_value = 0;
crc_value = crc_data_register_read();
```

crc_free_data_register_read

The description of crc_free_data_register_read is shown as below:

Table 3-87. Function crc_free_data_register_read

Function name	crc_free_data_register_read
Function prototype	uint8_t crc_free_data_register_read(void);
Function descriptions	read the value of the free data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	8-bit value of the free data register (0-0xFF)

Example:

```
/* read crc free data register */
uint8_t crc_value = 0;
crc_value = crc_free_data_register_read();
```

crc_free_data_register_write

The description of crc_free_data_register_write is shown as below:

Table 3-88. Function crc_free_data_register_write

Function name	crc_free_data_register_write
Function prototype	void crc_free_data_register_write(uint8_t free_data);
Function descriptions	write data to the free data register
Precondition	-
The called functions	-
Input parameter{in}	
free_data	specify 8-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write the free data register */
crc_free_data_register_write(0x11);
```

crc_single_data_calculate

The description of crc_single_data_calculate is shown as below:

Table 3-89. Function crc_single_data_calculate

Function name	crc_single_data_calculate
Function prototype	uint32_t crc_single_data_calculate(uint32_t sdata);
Function descriptions	calculate the CRC value of a 32-bit data
Precondition	-
The called functions	-
Input parameter{in}	
sdata	specify 32-bit data
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

```
/* CRC calculate a 32-bit data */
uint32_t val = 0, valcrc = 0;
val = (uint32_t)0xabcd1234;
rcu_periph_clock_enable(RCU_CRC);
valcrc = crc_single_data_calculate(val);
```

crc_block_data_calculate

The description of crc_block_data_calculate is shown as below:

Table 3-90. Function crc_block_data_calculate

Function name	crc_block_data_calculate
Function prototype	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
Function descriptions	calculate the CRC value of an array of 32-bit values
Precondition	-
The called functions	-
Input parameter{in}	
array	pointer to an array of 32 bits data words
Input parameter{in}	
size	size of the array
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

```
/* CRC calculate a 32-bit data array */

#define BUFFER_SIZE          6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_block_data_calculate((uint32_t *) data_buffer, BUFFER_SIZE);
```

3.5. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.5.1](#). the DBG firmware functions are introduced in chapter [3.5.2](#).

3.5.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

Table 3-91. DBG Registers

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register0
DBG_CTL1	DBG control register1
DBG_CTL2	DBG control register2

3.5.2. Descriptions of Peripheral functions

DBG firmware functions are listed in the table shown as below:

Table 3-92. DBG firmware function

Function name	Function description
dbg_deinit	deinitialize the DBG
dbg_id_get	read DBG_ID code register
dbg_low_power_enable	enable low power behavior when the MCU is in debug mode
dbg_low_power_disable	disable low power behavior when the MCU is in debug mode
dbg_periph_enable	enable peripheral behavior when the MCU is in debug mode
dbg_periph_disable	disable peripheral behavior when the MCU is in debug mode

Enum dbg_periph_enum

Table 3-93. Enum dbg_periph_enum

Member name	Function description
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_RTC_HOLD	hold RTC calendar and wakeup counter when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted
DBG_TIMER16_HOLD	hold TIMER16 counter when core is halted

dbg_deinit

The description of dbg_deinit is shown as below:

Table 3-94. Function dbg_deinit

Function name	dbg_deinit
Function prototype	void dbg_deinit (void);
Function descriptions	deinitialize the DBG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the DBG*/
dbg_deinit();
```

dbg_id_get

The description of dbg_id_get is shown as below:

Table 3-95. Function dbg_id_get

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);

Function descriptions	read DBG_ID code register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	DBG_ID code (0-0xFFFFFFFF)

Example:

```
/* read DBG_ID code register */
```

```
uint32_t id_value = 0;
```

```
id_value = dbg_id_get();
```

dbg_low_power_enable

The description of dbg_low_power_enable is shown as below:

Table 3-96. Function dbg_low_power_enable

Function name	dbg_low_power_enable
Function prototype	void dbg_low_power_enable(uint32_t dbg_low_power);
Function descriptions	enable low power behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_low_power	low power mode
<i>DBG_LOW_POWER_SLEEP</i>	keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low power behavior when the mcu is in debug mode */
```

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

dbg_low_power_disable

The description of dbg_low_power_disable is shown as below:

Table 3-97. Function dbg_low_power_disable

Function name	dbg_low_power_disable
Function prototype	void dbg_low_power_disable(uint32_t dbg_low_power);
Function descriptions	disable low power behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_low_power	low power mode
<i>DBG_LOW_POWER_SLEEP</i>	keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable low power behavior when the mcu is in debug mode */
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

dbg_periph_enable

The description of dbg_periph_enable is shown as below:

Table 3-98. Function dbg_periph_enable

Function name	dbg_periph_enable
Function prototype	void dbg_periph_enable(dbg_periph_enum dbg_periph);
Function descriptions	enable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	Peripheral refer to Table 3-93. Enum dbg_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable peripheral behavior when the mcu is in debug mode */
dbg_periph_enable(DBG_TIMER0_HOLD);
```

dbg_periph_disable

The description of dbg_periph_disable is shown as below:

Table 3-99. Function dbg_periph_disable

Function name	dbg_periph_disable
Function prototype	void dbg_periph_disable(dbg_periph_enum dbg_periph);
Function descriptions	disable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	peripheral refer to Table 3-93. Enum dbg_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable peripheral behavior when the mcu is in debug mode */
dbg_periph_disable(DBG_TIMER0_HOLD);
```

3.6. DMA

The direct memory access (DMA) controller provides a hardware method of transferring data between peripherals and/or memory without intervention from the CPU, thereby freeing up bandwidth for other system functions. The DMA registers are listed in chapter [3.6.1](#), the DMA firmware functions are introduced in chapter [3.6.2](#).

3.6.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

Table 3-100. DMA Registers

Registers	Descriptions
DMA_INTF0	Interrupt flag register 0
DMA_INTF1	Interrupt flag register 1
DMA_INTC0	Interrupt flag clear register 0
DMA_INTC1	Interrupt flag clear register 1

Registers	Descriptions
DMA_CHxCTL (x=0..7)	Channel x control register
DMA_CHxCNT (x=0..7)	Channel x counter register
DMA_CHxPADDR (x=0..7)	Channel x peripheral base address register
DMA_CHxM0ADDR (x=0..7)	Channel x memory 0 base address register
DMA_CHxM1ADDR (x=0..7)	Channel x memory 1 base address register
DMA_CHxFCTL (x=0..7)	DMA channel x FIFO control register

3.6.2. Descriptions of Peripheral functions

DMA firmware functions are listed in the table shown as below:

Table 3-101. DMA firmware function

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_single_struct_para_struct_init	initialize the DMA single data mode parameters struct with the default values
dma_multi_struct_para_struct_init	initialize the DMA multi data mode parameters struct with the default values
dma_single_data_mode_init	DMA single data mode initialize
dma_multi_data_mode_init	DMA multi data mode initialize
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_burst_beats_config	configure transfer burst beats of memory
dma_periph_burst_beats_config	configure transfer burst beats of peripheral
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_address_generation_config	configure next address increasement algorithm of memory
dma_peripheral_address_generation_config	configure next address increasement algorithm of peripheral
dma_circulation_enable	enable DMA circulation mode

Function name	Function description
dma_circulation_disable	disable DMA circulation mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_switch_buffer_mode_config	configure DMA switch buffer mode
dma_using_memory_get	get DMA using memory
dma_channel_subperipheral_select	select DMA channel peripheral
dma_flow_controller_config	configure DMA flow controller
dma_switch_buffer_mode_enable	enable DMA switch buffer mode
dma_switch_buffer_mode_disable	disable DMA switch buffer mode
dma_fifo_status_get	get DMA FIFO status
dma_flag_get	check the designated flag of a DMA channel is set or not
dma_flag_clear	clear the designated flag of a DMA channel
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt
dma_interrupt_flag_get	check the designated interrupt flag of a DMA channel is set or not
dma_interrupt_flag_clear	clear the designated interrupt flag of a DMA channel

Enum dma_channel_enum

Table 3-102. Enum dma_channel_enum

Member name	Function description
DMA_CH0	DMA Channel0
DMA_CH1	DMA Channel1
DMA_CH2	DMA Channel2
DMA_CH3	DMA Channel3
DMA_CH4	DMA Channel4
DMA_CH5	DMA Channel5
DMA_CH6	DMA Channel6
DMA_CH7	DMA Channel7

Enum dma_subperipheral_enum

Table 3-103. Enum dma_subperipheral_enum

Member name	Function description
DMA_SUBPERI0	DMA Peripheral 0
DMA_SUBPERI1	DMA Peripheral 1
DMA_SUBPERI2	DMA Peripheral 2
DMA_SUBPERI3	DMA Peripheral 3
DMA_SUBPERI4	DMA Peripheral 4
DMA_SUBPERI5	DMA Peripheral 5

DMA_SUBPERI6	DMA Peripheral 6
DMA_SUBPERI7	DMA Peripheral 7

Structure dma_multi_data_parameter_struct

Table 3-104. Structure dma_multi_data_parameter_struct

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_width	transfer data size of memory
memory_inc	memory increasing mode
memory_burst_width	multi data mode enable
periph_burst_width	multi data mode enable
critical_value	FIFO critical
circular_mode	DMA circular mode
direction	channel data transfer direction
number	channel transfer number
priority	channel priority level

Structure dma_single_data_parameter_struct

Table 3-105. Structure dma_single_data_parameter_struct

Member name	Function description
periph_addr	peripheral base address
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_inc	memory increasing mode
periph_memory_width	transfer data size of peripheral
circular_mode	DMA circular mode
direction	channel data transfer direction
number	channel transfer number
priority	channel priority level

dma_deinit

The description of dma_deinit is shown as below:

Table 3-106. Function dma_deinit

Function name	dma_deinit
Function prototype	void dma_deinit(dma_channel_enum channelx);

Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA channel0 deinitialize*/
dma_deinit(DMA_CH0);
```

dma_single_data_para_struct_init

The description of dma_single_data_para_struct_init is shown as below:

Table 3-107. Function dma_single_data_para_struct_init

Function name	dma_single_data_para_struct_init
Function prototype	void dma_single_data_para_struct_init(dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize the DMA single data mode parameters struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_struct	the address of structure for initialization, the structure members can refer to Table 3-105. Structure dma_single_data_parameter_struct
Return value	
-	-

Example:

```
/* initialize the parameters struct of DMA with single data mode*/
dma_single_data_parameter_struct dma_init_struct;
dma_single_data_para_struct_init(&dma_init_struct);
```

dma_multi_data_para_struct_init

The description of dma_multi_data_para_struct_init is shown as below:

Table 3-108. Function dma_multi_data_para_struct_init

Function name	dma_multi_data_para_struct_init
Function prototype	void dma_multi_data_para_struct_init(dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize the DMA multi data mode parameters struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_struct	the address of structure for initialization, the structure members can refer to Table 3-104. Structure dma_multi_data_parameter_struct
Return value	
-	-

Example:

```
/* initialize the parameters struct of DMA with multi data mode*/
dma_multi_data_parameter_struct dma_init_struct;
dma_multi_data_para_struct_init (&dma_init_struct);
```

dma_single_data_mode_init

The description of dma_single_data_mode_init is shown as below:

Table 3-109. Function dma_single_data_mode_init

Function name	dma_single_data_mode_init
Function prototype	void dma_single_data_mode_init(dma_channel_enum channelx, dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize DMA single data mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
init_struct	the address of structure for initialization, the structure members can refer to Table 3-105. Structure dma_single_data_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* DMA channel0 single data mode initialize */

dma_single_data_parameter_struct dma_init_struct;

dma_deinit(DMA_CH0);

dma_single_data_para_struct_init(&dma_init_struct);


dma_init_struct.direction = DMA_MEMORY_TO_MEMORY;

dma_init_struct.memory0_addr = (uint32_t)g_destbuf;

dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;

dma_init_struct.periph_memory_width = DMA_MEMORY_WIDTH_32BIT;

dma_init_struct.number = TRANSFER_NUM;

dma_init_struct.circular_mode = DMA_CIRCULAR_MODE_DISABLE;

dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;

dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;

dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;

dma_single_data_mode_init(DMA1, DMA_CH0, &dma_init_struct);

```

dma_multi_data_mode_init

The description of dma_multi_data_mode_init is shown as below:

Table 3-110. Function dma_multi_data_mode_init

Function name	dma_multi_data_mode_init
Function prototype	void dma_multi_data_mode_init(dma_channel_enum channelx, dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize DMA multi data mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
init_struct	the address of structure for initialization, the structure members can refer to Table 3-104. Structure dma_multi_data_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* DMA channel0 multi data mode initialize */

dma_multi_data_parameter_struct dma_init_parameter;

dma_multi_data_para_struct_init(&dma_init_parameter);

dma_deinit(DMA_CH0);

dma_init_parameter.periph_addr = (uint32_t)source;

dma_init_parameter.periph_width = DMA_PERIPH_WIDTH_16BIT;

dma_init_parameter.periph_inc = DMA_PERIPH_INCREASE_ENABLE;

dma_init_parameter.memory0_addr = (uint32_t)destination;

dma_init_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;

dma_init_parameter.memory_inc = DMA_MEMORY_INCREASE_ENABLE;

dma_init_parameter.memory_burst_width = DMA_MEMORY_BURST_4_BEAT;

dma_init_parameter.periph_burst_width = DMA_PERIPH_BURST_4_BEAT;

dma_init_parameter.critical_value = DMA_FIFO_2_WORD;

dma_init_parameter.circular_mode = DMA_CIRCULAR_MODE_DISABLE;

dma_init_parameter.direction = DMA_MEMORY_TO_MEMORY;

dma_init_parameter.number = BUFFER_SIZE;

dma_init_parameter.priority = DMA_PRIORITY_ULTRA_HIGH;

dma_multi_data_mode_init(DMA1,DMA_CH0, &dma_init_parameter);

```

dma_periph_address_config

The description of dma_periph_address_config is shown as below:

Table 3-111. Function dma_periph_address_config

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum

Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 peripheral address */

#define USART_DATA_ADDR (USART0 + 0x00000024U)

dma_periph_address_config (DMA_CH0, USART_DATA_ADDR);
```

dma_memory_address_config

The description of dma_memory_address_config is shown as below:

Table 3-112. Function dma_memory_address_config

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(dma_channel_enum channelx, uint8_t memory_flag, uint32_t address);
Function descriptions	set DMA memory0 base address
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
address	memory0 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 memory0 address */

uint32_t array[10] = {0};

dma_memory_address_config (DMA_CH0, array);
```

dma_transfer_number_config

The description of dma_transfer_number_config is shown as below:

Table 3-113. Function dma_transfer_number_config

Function name	dma_transfer_number_config
Function prototype	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
Function descriptions	set the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
number	data transfer number (0x00000000 – 0x0000FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer number */
#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

dma_transfer_number_get

The description of dma_transfer_number_get is shown as below:

Table 3-114. Function dma_transfer_number_get

Function name	dma_transfer_number_get
Function prototype	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
Function descriptions	get the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
uint32_t	0x00000000 – 0x0000FFFF

Example:

```
/* get channel0 memory0 address */
```

```
uint32_t number = 0;
```

```
number = dma_transfer_number_get(DMA_CH0);
```

dma_priority_config

The description of dma_priority_config is shown as below:

Table 3-115. Function dma_priority_config

Function name	dma_priority_config
Function prototype	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
Function descriptions	configure priority level of DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
priority	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 priority */
```

```
dma_priority_config(DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

dma_memory_burst_beats_config

The description of dma_memory_burst_beats_config is shown as below:

Table 3-116. Function dma_memory_burst_beats_config

Function name	dma_memory_burst_beats_config
Function prototype	void dma_memory_burst_beats_config(dma_channel_enum channelx, uint32_t mbeat);
Function descriptions	configure transfer burst beats of memory
Precondition	-

The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
mbeat	transfer burst beats
<i>DMA_MEMORY_BURST_SINGLE</i>	memory transfer single burst
<i>DMA_MEMORY_BURST_4_BEAT</i>	memory transfer 4-beat burst
<i>DMA_MEMORY_BURST_8_BEAT</i>	memory transfer 8-beat burst
<i>DMA_MEMORY_BURST_16_BEAT</i>	memory transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer burst beats of memory */
```

```
dma_memory_burst_beats_config(DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

dma_periph_burst_beats_config

The description of dma_periph_burst_beats_config is shown as below:

Table 3-117. Function dma_periph_burst_beats_config

Function name	dma_periph_burst_beats_config
Function prototype	void dma_periph_burst_beats_config (dma_channel_enum channelx, uint32_t pbeat);
Function descriptions	configure transfer burst beats of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
pbeat	transfer burst beats
<i>DMA_PERIPH_BURST_SINGLE</i>	peripheral transfer single burst
<i>DMA_PERIPH_BURST_4_BEAT</i>	peripheral transfer 4-beat burst

<i>DMA_PERIPH_BURST_8_BEAT</i>	peripheral transfer 8-beat burst
<i>DMA_PERIPH_BURST_16_BEAT</i>	peripheral transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer burst beats of peripheral */
```

```
dma_periph_burst_beats_config(DMA_CH0, DMA_PERIPH_BURST_8_BEAT);
```

dma_memory_width_config

The description of dma_memory_width_config is shown as below:

Table 3-118. Function dma_memory_width_config

Function name	dma_memory_width_config
Function prototype	void dma_memory_width_config(dma_channel_enum channelx, uint32_t msize);
Function descriptions	configure transfer data size of memory
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
msize	transfer data width of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDTH_32BIT</i>	transfer data width of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer memory width */
```

```
dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

dma_periph_width_config

The description of dma_periph_width_config is shown as below:

Table 3-119. Function dma_periph_width_config

Function name	dma_periph_width_config
Function prototype	void dma_periph_width_config (dma_channel_enum channelx, uint32_t psize);
Function descriptions	configure transfer data size of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
psize	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer peripheral width */
dma_periph_width_config(DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

dma_memory_address_generation_config

The description of dma_memory_address_generation_config is shown as below:

Table 3-120. Function dma_memory_address_generation_config

Function name	dma_memory_address_generation_config
Function prototype	void dma_memory_address_generation_config(dma_channel_enum channelx, uint8_t generation_algorithm);
Function descriptions	configure memory address generation algorithm
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel

<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
generation_algorithm	memory address generation algorithm
<i>DMA_MEMORY_INCR EASE_ENABLE</i>	next address of memory is increasing address mode
<i>DMA_MEMORY_INCR EASE_DISABLE</i>	next address of memory is fixed address mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 memory address generation algorithm as memory increase*/
```

```
dma_memory_address_generation_config(DMA_CH0, DMA_MEMORY_INCREASE_ENABLE);
```

dma_peripheral_address_generation_config

The description of dma_peripheral_address_generation_config is shown as below:

Table 3-121. Function dma_peripheral_address_generation_config

Function name	dma_peripheral_address_generation_config
Function prototype	void dma_peripheral_address_generation_config(dma_channel_enum channelx, uint8_t generation_algorithm);
Function descriptions	configure peripheral address generation algorithm
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
generation_algorithm	transfer data width of peripheral
<i>DMA_PERIPH_INCREASE ASE_ENABLE</i>	next address of peripheral is increasing address mode
<i>DMA_PERIPH_INCREASE ASE_DISABLE</i>	next address of peripheral is fixed address mode
<i>DMA_PERIPH_INCREASE ASE_FIX</i>	increasing steps of peripheral address is fixed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 peripheral address generation algorithm as peripheral increase*/
```

```
dma_peripheral_address_generation_config(DMA_CH0, DMA_PERIPH_INCREASE_ENABLE);
```

dma_circulation_enable

The description of dma_circulation_enable is shown as below:

Table 3-122. Function dma_circulation_enable

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel0 circulation mode */
```

```
dma_circulation_enable(DMA_CH0);
```

dma_circulation_disable

The description of dma_circulation_disable is shown as below:

Table 3-123. Function dma_circulation_disable

Function name	dma_circulation_disable
Function prototype	void dma_circulation_disable(dma_channel_enum channelx);
Function descriptions	disable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* disable DMA channel0 circulation mode */
```

```
dma_circulation_disable(DMA_CH0);
```

dma_channel_enable

The description of dma_channel_enable is shown as below:

Table 3-124. Function dma_channel_enable

Function name	dma_channel_enable
Function prototype	void dma_channel_enable(dma_channel_enum channelx);
Function descriptions	enable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel0 */
```

```
dma_channel_enable(DMA_CH0);
```

dma_channel_disable

The description of dma_channel_disable is shown as below:

Table 3-125. Function dma_channel_disable

Function name	dma_channel_disable
Function prototype	void dma_channel_disable(dma_channel_enum channelx);
Function descriptions	disable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 */
```

```
dma_channel_disable(DMA_CH0);
```

dma_transfer_direction_config

The description of dma_transfer_direction_config is shown as below:

Table 3-126. Function dma_transfer_direction_config

Function name	dma_transfer_direction_config
Function prototype	void dma_transfer_direction_config(dma_channel_enum channelx, uint8_t direction);
Function descriptions	configure the direction of data transfer on the channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
direction	specify the direction of data transfer
<i>DMA_PERIPHERAL_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPHERAL</i>	read from memory and write to peripheral
<i>DMA_MEMORY_TO_MEMORY</i>	read from memory and write to memory
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 transfer direction as peripheral to memory mode*/
```

```
dma_transfer_direction_config(DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

dma_switch_buffer_mode_config

The description of dma_switch_buffer_mode_config is shown as below:

Table 3-127. Function dma_switch_buffer_mode_config

Function name	dma_switch_buffer_mode_config
Function prototype	void dma_switch_buffer_mode_config(dma_channel_enum channelx, uint32_t memory1_addr, uint32_t memory_select);
Function descriptions	configure DMA switch buffer mode
Precondition	-

The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
memory1_addr	memory1 base address
Input parameter{in}	
memory_select	select memory transfer area
DMA_MEMORY_0	select memory0 as memory transfer area
DMA_MEMORY_1	select memory1 as memory transfer area
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 switch buffer mode*/
```

```
uint32_t mem[32] = {0};
```

```
dma_switch_buffer_mode_config(DMA_CH0, mem, DMA_MEMORY_0);
```

dma_using_memory_get

The description of dma_using_memory_get is shown as below:

Table 3-128. Function dma_using_memory_get

Function name	dma_using_memory_get
Function prototype	uint32_t dma_using_memory_get(dma_channel_enum channelx);
Function descriptions	configure DMA switch buffer mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
uint32_t	the memory buffer area currently used
DMA_MEMORY_0	memory0 is selected as memory transfer area
DMA_MEMORY_1	memory1 is selected as memory transfer area

Example:

```
/* get DMA channel0 transfer buffer currently used */
```

```
uint32_t buf_addr = DMA_MEMORY_0;
```

```
buf_addr = dma_using_memory_get(DMA_CH0);
```

dma_channel_subperipheral_select

The description of dma_channel_subperipheral_select is shown as below:

Table 3-129. Function dma_channel_subperipheral_select

Function name	dma_channel_subperipheral_select
Function prototype	void dma_channel_subperipheral_select(dma_channel_enum channelx, dma_subperipheral_enum sub_periph);
Function descriptions	DMA channel peripheral select
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
sub_periph	specify DMA channel peripheral
<i>DMA_SUBPERIx(x=0..7)</i>	DMA Peripheral x(x = 0...7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 peripheral */
```

```
dma_channel_subperipheral_select(DMA_CH0, DMA_SUBPERI1);
```

dma_flow_controller_config

The description of dma_flow_controller_config is shown as below:

Table 3-130. Function dma_flow_controller_config

Function name	dma_flow_controller_config
Function prototype	void dma_flow_controller_config(dma_channel_enum channelx, uint32_t controller);
Function descriptions	configure DMA flow controller
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel

<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
controller	specify DMA flow controler
<i>DMA_FLOW_CONTROLLER_DMA</i>	DMA is the flow controller
<i>DMA_FLOW_CONTROLLER_PERI</i>	peripheral is the flow controller
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 flow controller */
```

```
dma_flow_controller_config(DMA_CH0, DMA_FLOW_CONTROLLER_DMA);
```

dma_switch_buffer_mode_enable

The description of dma_switch_buffer_mode_enable is shown as below:

Table 3-131. Function dma_switch_buffer_mode_enable

Function name	dma_switch_buffer_mode_enable
Function prototype	void dma_switch_buffer_mode_enable(dma_channel_enum channelx);
Function descriptions	enable DMA switch buffer mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel0 switch buffer mode */
```

```
dma_switch_buffer_mode_enable (DMA_CH0);
```

dma_switch_buffer_mode_disable

The description of dma_switch_buffer_mode_disable is shown as below:

Table 3-132. Function dma_switch_buffer_mode_enable

Function name	dma_switch_buffer_mode_disable
Function prototype	void dma_switch_buffer_mode_disable(dma_channel_enum channelx);
Function descriptions	enable DMA switch buffer mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 switch buffer mode */
dma_switch_buffer_mode_disable (DMA_CH0);
```

dma_fifo_status_get

The description of dma_fifo_status_get is shown as below:

Table 3-133. Function dma_fifo_status_get

Function name	dma_fifo_status_get
Function prototype	uint32_t dma_fifo_status_get(dma_channel_enum channelx);
Function descriptions	get DMA FIFO status
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Output parameter{out}	
-	-
Return value	
uint32_t	the using memory in FIFO

Example:

```
/* get DMA channel0 FIFO status */
uint32_t fifo_state = 0;
fifo_state = dma_fifo_status_get(DMA_CH0);
```

dma_flag_get

The description of dma_flag_get is shown as below:

Table 3-134. Function dma_flag_get

Function name	dma_flag_get
Function prototype	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
Function descriptions	check the designated flag of a DMA channel is set or not
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transfer finish flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check DMA channel0 full transfer finish flag set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_flag_get(DMA_CH0, DMA_FLAG_FTF);
```

dma_flag_clear

The description of dma_flag_clear is shown as below:

Table 3-135. Function dma_flag_clear

Function name	dma_flag_clear
Function prototype	void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);
Function descriptions	clear the designated flag of a DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum

Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transfer finish flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA channel0 full transfer finish flag */
dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

dma_interrupt_enable

The description of dma_interrupt_enable is shown as below:

Table 3-136. Function dma_interrupt_enable

Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	enable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
source	DMA interrupt source
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable
<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* enable DMA channel0 full transfer finish interrupt */
```

```
dma_interrupt_enable(DMA_CH0, DMA_INT_FTF);
```

dma_interrupt_disable

The description of dma_interrupt_disable is shown as below:

Table 3-137. Function dma_interrupt_disable

Function name	dma_interrupt_disable
Function prototype	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	disable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
source	DMA interrupt source
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable
<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 full transfer finish interrupt */
```

```
dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

dma_interrupt_flag_get

The description of dma_interrupt_flag_get is shown as below:

Table 3-138. Function dma_interrupt_flag_get

Function name	dma_interrupt_flag_get
Function prototype	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t int_flag);
Function descriptions	check the designated interrupt flag of a DMA channel is set or not
Precondition	-

The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
int_flag	specify get which interrupt flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception interrupt flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception interrupt flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check DMA channel0 full transfer finish interrupt flag set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_interrupt_flag_get (DMA_CH0, DMA_INT_FLAG_FTF);
```

dma_interrupt_flag_clear

The description of dma_interrupt_flag_clear is shown as below:

Table 3-139. Function dma_interrupt_flag_clear

Function name	dma_interrupt_flag_clear
Function prototype	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t int_flag);
Function descriptions	clear the designated interrupt flag of a DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to Table 3-102. Enum dma_channel_enum
Input parameter{in}	
int_flag	specify get which interrupt flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception interrupt flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception interrupt flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA channel0 full transfer finish interrupt flag */
```

```
dma_interrupt_flag_clear(DMA_CH0, DMA_INT_FLAG_FTF);
```

3.7. ECLIC

RISC-V integrates the Enhancement Core-Local Interrupt Controller (ECLIC) for efficient interrupts processing. ECLIC is designed to provide low-latency, vectored, pre-emptive interrupts for RISC-V systems. The ECLIC firmware functions are introduced in chapter [3.7.1](#).

3.7.1. Descriptions of Peripheral functions

ECLIC firmware functions are listed in the table shown as below:

Table 3-140. ECLIC firmware function

Function name	Function description
eclic_global_interrupt_enable	enable the global interrupt
eclic_global_interrupt_disable	disable the global interrupt
eclic_level_threshold_set	set machine mode interrupt level threshold
eclic_priority_group_set	set the priority group
eclic_irq_enable	enable the interrupt request
eclic_irq_disable	disable the interrupt request
eclic_system_reset	reset system

Enum IRQn_Type

Table3-141. Enum IRQn_Type

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	tamper and timestamp through EXTI line detect
RTC_WKUP_IRQn	RTC wakeup through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_IRQn	RCU interrupt
EXTI0_IRQn	EXTI line 0 interrupts

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA_Channel0_IRQn	DMA0 channel0 interrupt
DMA_Channel1_IRQn	DMA0 channel1 interrupt
DMA_Channel2_IRQn	DMA0 channel2 interrupt
DMA_Channel3_IRQn	DMA0 channel3 interrupt
DMA_Channel4_IRQn	DMA0 channel4 interrupt
DMA_Channel5_IRQn	DMA0 channel5 interrupt
DMA_Channel6_IRQn	DMA0 channel6 interrupt
DMA_Channel7_IRQn	DMA0 channel7 interrupt
ADC_IRQn	ADC interrupt
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_IRQn	TIMER0 break interrupts
TIMER0_UP_IRQn	TIMER0 update interrupts
TIMER0_CMT_IRQn	TIMER0 commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupts
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI_IRQn	SPI interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt
VLVDF_IRQn	VLVDF interrupt
TIMER15_IRQn	TIMER15 global interrupt
TIMER16_IRQn	TIMER16 global interrupt
I2C0_WKUP_IRQn	I2C0 wakeup interrupt
USART0_WKUP_IRQn	USART0 wakeup interrupt
TIMER5_IRQn	TIMER5 global interrupt
WIFI_TRIGGER_IRQn	WIFI protocol trigger interrupt
WIFI_MAC_IRQn	WIFI MAC interrupt
WIFI_TX_IRQn	WIFI Tx interrupt

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
WIFI_RX_IRQn	WIFI Rx interrupt
LA_IRQn	LA interrupt
WIFI_WKUP_IRQn	WIFI wakeup interrupt
BLE_WKUP_IRQn	BLE wakeup interrupt
PLATFORM_WAKE_IRQn	Platform wake interrupt
ISO_BT_STAMP0_IRQn	ISO Bluetooth TimeStamp interrupt0
ISO_BT_STAMP1_IRQn	ISO Bluetooth TimeStamp interrupt1
ISO_BT_STAMP2_IRQn	ISO Bluetooth TimeStamp interrupt2
ISO_BT_STAMP3_IRQn	ISO Bluetooth TimeStamp interrupt3
ISO_BT_STAMP4_IRQn	ISO Bluetooth TimeStamp interrupt4
ISO_BT_STAMP5_IRQn	ISO Bluetooth TimeStamp interrupt5
ISO_BT_STAMP6_IRQn	ISO Bluetooth TimeStamp interrupt6
ISO_BT_STAMP7_IRQn	ISO Bluetooth TimeStamp interrupt7
PMU_IRQn	PMU interrupt
CAU_IRQn	CAU interrupt
HAU_TRNG_IRQn	HAU and TRNG interrupt
WIFI_INT_IRQn	WIFI global interrupt
WIFI_SW_TRIG_IRQn	SW triggered interrupt
WIFI_FINE_TIMER_TARGET_IRQn	Fine timer target interrupt
WIFI_STAMP_TARGET1_IRQn	Timestamp target 1 interrupt
WIFI_STAMP_TARGET2_IRQn	Timestamp target 2 interrupt
WIFI_STAMP_TARGET3_IRQn	Timestamp target 3 interrupt
WIFI_ENCRYPTION_ENGINE_IRQn	Encryption engine interrupt
WIFI_SLEEP_MODE_IRQn	Sleep mode interrupt
WIFI_HALF_SLOT_IRQn	Half slot interrupt
WIFI_FIFO_ACTIVITY_IRQn	FIFO activity interrupt
WIFI_ERROR_IRQn	Error interrupt
WIFI_FREQ_SELECT_IRQn	Frequency selection interrupt
EFUSE_IRQn	EFUSE global interrupt
QSPI_IRQn	QSPI global interrupt
PKCAU_IRQn	PKCAU global interrupt

eclic_global_interrupt_enable

The description of eclic_global_interrupt_enable is shown as below:

Table 3-142. Function eclic_global_interrupt_enable

Function name	eclic_global_interrupt_enable
Function prototype	void eclic_global_interrupt_enable(void);
Function descriptions	enable the global interrupt

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the global interrupt */
eclic_global_interrupt_enable();
```

eclic_global_interrupt_disable

The description of eclic_global_interrupt_disable is shown as below:

Table 3-143. Function eclic_global_interrupt_disable

Function name	eclic_global_interrupt_disable
Function prototype	void eclic_global_interrupt_disable(void);
Function descriptions	disable the global interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the global interrupt */
eclic_global_interrupt_disable();
```

eclic_level_threshold_set

The description of eclic_level_threshold_set is shown as below:

Table 3-144. Function eclic_level_threshold_set

Function name	eclic_level_threshold_set
Function prototype	void eclic_level_threshold_set(uint8_t threshold);
Function descriptions	set machine mode interrupt level threshold
Precondition	-
The called functions	-

Input parameter{in}	
threshold	the level threshold needed to set (0 ~ 15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set machine mode interrupt level threshold */
eclic_level_threshold_set(0);
```

eclic_priority_group_set

The description of eclic_priority_group_set is shown as below:

Table 3-145. Function eclic_priority_group_set

Function name	eclic_priority_group_set
Function prototype	void eclic_priority_group_set(uint8_t prigroup);
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
prigroup	specify the priority group
<i>ECLIC_PRIGROUP_LEV EL0_Prio4</i>	0 bits for level, 4 bits for priority
<i>ECLIC_PRIGROUP_LEV EL1_Prio3</i>	1 bits for level, 3 bits for priority
<i>ECLIC_PRIGROUP_LEV EL2_Prio2</i>	2 bits for level, 2 bits for priority
<i>ECLIC_PRIGROUP_LEV EL3_Prio1</i>	3 bits for level, 1 bits for priority
<i>ECLIC_PRIGROUP_LEV EL4_Prio0</i>	4 bits for level, 0 bits for priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the priority group */
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_Prio4);
```

eclic_irq_enable

The description of eclic_irq_enable is shown as below:

Table 3-146. Function eclic_irq_enable

Function name	eclic_irq_enable
Function prototype	void eclic_irq_enable(IRQn_Type source, uint8_t level, uint8_t priority);
Function descriptions	enable the interrupt request
Precondition	-
The called functions	-
Input parameter{in}	
source	interrupt request, detailed in Table3-141. Enum IRQn_Type
Input parameter{in}	
level	the level needed to set (maximum is 15, refer to the priority group)
Input parameter{in}	
priority	the priority needed to set (maximum is 15, refer to the priority group)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable and set key EXTI interrupt to the specified priority */
eclic_global_interrupt_enable();
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL3_PRIO1);
eclic_irq_enable(EXTI10_15_IRQn, 1, 1);
```

eclic_irq_disable

The description of eclic_irq_disable is shown as below:

Table 3-147. Function eclic_irq_disable

Function name	eclic_irq_disable
Function prototype	void eclic_irq_disable(IRQn_Type source)
Function descriptions	disable the interrupt request
Precondition	-
The called functions	-
Input parameter{in}	
source	interrupt request, detailed in Table3-141. Enum IRQn_Type
Output parameter{out}	
-	-
Return value	
-	-

Example:


```

/* disable the interrupt request */
eclic_irq_disable(EXTI10_15_IRQn);

```

eclic_system_reset

The description of eclic_system_reset is shown as below:

Table 3-148. Function eclic_system_reset

Function name	eclic_system_reset
Function prototype	void eclic_system_reset(void);
Function descriptions	reset system
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* reset system */
eclic_system_reset();

```

3.8. EFUSE

The Efuse controller has efuse macro that store system paramters. As a non-volatile unit of storage, the bit of efuse macro cannot be restored to 0 once it is programmed to 1. The EFUSE registers are listed in chapter [3.8.1](#), the EFUSE firmware functions are introduced in chapter [3.8.2](#).

3.8.1. Descriptions of Peripheral registers

EFUSE registers are listed in the table shown as below:

Table 3-149. EFUSE Registers

Registers	Descriptions
EFUSE_CS	control and status register
EFUSE_ADDR	address register
EFUSE_CTL0	control 0 register
EFUSE_CTL1	control 1 register
EFUSE_FPCTL	flash protection control register
EFUSE_USERCTL	user control register
EFUSE_RESx(x =	mcu reserved register x(x = 0...2)

Registers	Descriptions
0...2)	
EFUSE_AESKEYx(x = 0...3)	firmware AES key register x(x = 0...3)
EFUSE_ROTTPKKE Yx(x = 0...7)	RoTPK key register x(x = 0...7)
EFUSE_PUIDx(x = 0...3)	product UID register x(x = 0...3)
EFUSE_HUKKEYx(x = 0...3)	HUK key register x(x = 0...3)
EFUSE_USER_DA TAX(x = 0...7)	user data register x(x = 0...7)
EFUSE_BOOTADD R	boot address register

3.8.2. Descriptions of Peripheral functions

EFUSE firmware functions are listed in the table shown as below:

Table 3-150. EFUSE firmware function

Function name	Function description
efuse_read	read EFUSE value
efuse_write	write EFUSE
efuse_boot_config	configure boot field
efuse_control1_config	configure efuse control1 field
efuse_fp_config	configure flash protection field
efuse_user_control_config	configure user control
efuse_res_write	write MCU reserved value
efuse_aes_key_write	write AES key
efuse_rotpk_key_write	write ROTPK key
efuse_user_data_write	write user data
efuse_res_read	read MCU reserved value
efuse_aes_key_read	read AES key
efuse_rotpk_key_read	read ROTPK key
efuse_puid_read	read puid
efuse_huk_key_read	read huk key
efuse_user_data_read	read user data
efuse_boot_address_get	get boot address information
efuse_lock_config	lock efuse filed
efuse_flag_get	check efuse flag is set or not
efuse_flag_clear	clear efuse pending flag
efuse_interrupt_enable	enable efuse interrupt
efuse_interrupt_disable	disable efuse interrupt

Function name	Function description
efuse_interrupt_flag_get	check efuse interrupt flag is set or not
efuse_interrupt_flag_clear	clear efuse pending interrupt flag

Enum efuse_flag_enum

Table 3-151. Enum efuse_flag_enum

Member name	Function description
EFUSE_PGIF	programming operation completion flag
EFUSE_RDIF	read operation completion flag
EFUSE_OVBERIF	overstep boundary error flag

Enum efuse_clear_flag_enum

Table 3-152. Enum efuse_clear_flag_enum

Member name	Function description
EFUSE_PGIC	clear programming operation completion flag
EFUSE_RDIC	clear read operation completion flag
EFUSE_OVBERIC	clear overstep boundary error flag

Enum efuse_int_enum

Table 3-153. Enum efuse_int_enum

Member name	Function description
EFUSE_INTEN_PG	programming operation completion interrupt
EFUSE_INTEN_RD	read operation completion interrupt
EFUSE_INTEN_OVBER R	overstep boundary error interrupt

Enum efuse_int_flag_enum

Table 3-154. Enum efuse_int_flag_enum

Member name	Function description
EFUSE_INT_PGIF	programming operation completion interrupt flag
EFUSE_INT_RDIF	read operation completion interrupt flag
EFUSE_INT_OVBERIF	overstep boundary error interrupt flag

Enum efuse_clear_int_flag_enum

Table 3-155. efuse_clear_int_flag_enum

Member name	Function description
EFUSE_INT_PGIC	clear programming operation completion interrupt flag
EFUSE_INT_RDIC	clear read operation completion interrupt flag
EFUSE_INT_OVBERIC	clear overstep boundary error interrupt flag

Enum efuse_reg_lock_enum

Table 3-156. Enum efuse_reg_lock_enum

Member name	Function description
EFUSE_BOOT_LOCK	EFUSE boot bits lock bit
EFUSE_ROTPKKEY_LOCK	EFUSE_ROTPKKEY register lock bit
EFUSE_USER_DATA_LOCK	EFUSE_USER_DATA register lock bit
EFUSE_AESKEY_LOCK	EFUSE_AESKEY register lock bit
EFUSE_FPCTL_USERCTL_LOCK	EFUSE_FPCTL and EFUSE_USERCTL register lock bit

efuse_read

The description of efuse_read is shown as below:

Table 3-157. Function efuse_read

Function name	efuse_read
Function prototype	ErrStatus efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
Function descriptions	read EFUSE value
Precondition	-
The called functions	-
Input parameter{in}	
ef_addr	EFUSE address (0x00 – 0x60)
Input parameter{in}	
size	size of EFUSE (0x01 – 0x20 bytes)
Output parameter{out}	
buf	the buffer for storing read-out EFUSE register value
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* read EFUSE USER DATA value */
uint32_t buffer[8] = {0};

ErrStatus flag = ERROR;

flag = efuse_read(0x60, 32, buffer);
```

efuse_write

The description of efuse_write is shown as below:

Table 3-158. Function efuse_write

Function name	efuse_write
Function prototype	ErrStatus efuse_write(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
Function descriptions	write EFUSE
Precondition	-
The called functions	-
Input parameter{in}	
ef_addr	EFUSE address (0x00 – 0x60), when ef_addr is greater than 4, the ef_addr must be an integral multiple of 4
Input parameter{in}	
size	size of EFUSE (0x01 – 0x20 bytes)
Output parameter{out}	
buf	the buffer of data written to EFUSE
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write EFUSE USER DATA*/

uint32_t buffer[2] = {0x11223344, 0x55667788};

ErrStatus flag = ERROR;

flag = efuse_write(0x60, 8, buffer);
```

efuse_boot_config

The description of efuse_boot_config is shown as below:

Table 3-159. Function efuse_boot_config

Function name	efuse_boot_config
Function prototype	ErrStatus efuse_boot_config(uint32_t size, uint8_t bt_value[]);
Function descriptions	configure boot
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 1
Input parameter{in}	
bt_value	the value of boot configuration
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write Efuse control value */
```

```
uint8_t bt_value[1] = 0x32;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_boot_config(1, bt_value);
```

efuse_control1_config

The description of efuse_control1_config is shown as below:

Table 3-160. Function efuse_control1_config

Function name	efuse_control1_config
Function prototype	ErrStatus efuse_control1_config(uint32_t size, uint8_t ctl[]);
Function descriptions	configure efuse control1
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 1
Input parameter{in}	
ctl	control1 value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write control1 value */
```

```
uint8_t ctl[1] = 0x02;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_control1_config(1, ctl);
```

efuse_fp_config

The description of efuse_fp_config is shown as below:

Table 3-161. Function efuse_fp_config

Function name	efuse_fp_config
Function prototype	ErrStatus efuse_fp_config(uint32_t size, uint8_t fp_value[]);
Function descriptions	flash protection configuration
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 1

Input parameter{in}	
fp_value	flash protection value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write Flash protection value */

uint8_t fp_value[1] = 0x01;

ErrStatus flag = ERROR;

flag = efuse_fp_config(1, fp_value);
```

efuse_user_control_config

The description of efuse_user_control_config is shown as below:

Table 3-162. Function efuse_user_control_config

Function name	efuse_user_control_config
Function prototype	ErrStatus efuse_user_control_config(uint32_t size, uint8_t user_ctl[]);
Function descriptions	configure user control
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 1
Input parameter{in}	
user_ctl	user control value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure user control */

uint8_t user_ctl[1] = 0x02;

ErrStatus flag = ERROR;

flag = efuse_user_control_config(1, user_ctl);
```

efuse_res_write

The description of efuse_res_write is shown as below:

Table 3-163. Function efuse_res_write

Function name	efuse_res_write
Function prototype	ErrStatus efuse_res_write(uint32_t size, uint8_t buf[]);
Function descriptions	write MCU reserved value
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 12
Input parameter{in}	
buf	MCU reserved value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write MCU reserved value */
uint32_t buffer[3] = {0x11223344, 0x55667788, 0x9900aabb};
ErrStatus flag = ERROR;
flag = efuse_res_write(12, (uint8_t *)buffer);
```

efuse_aes_key_write

The description of efuse_aes_key_write is shown as below:

Table 3-164. Function efuse_aes_key_write

Function name	efuse_aes_key_write
Function prototype	ErrStatus efuse_aes_key_write(uint32_t size, uint8_t buf[]);
Function descriptions	write AES key
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 16
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write AES key value */
```



```
uint32_t buffer[4] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_aes_key_write(4, (uint8_t *)buffer);
```

efuse_rotpk_key_write

The description of efuse_rotpk_key_write is shown as below:

Table 3-165. Function efuse_rotpk_key_write

Function name	efuse_rotpk_key_write
Function prototype	ErrStatus efuse_rotpk_key_write(uint8_t buf[]);
Function descriptions	write ROTPK key
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 32
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write AES key value */
```

```
uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff  
0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_rotpk_key_write(32, (uint8_t *)buffer);
```

efuse_user_data_write

The description of efuse_user_data_write is shown as below:

Table 3-166. Function efuse_user_data_write

Function name	efuse_user_data_write
Function prototype	ErrStatus efuse_user_data_write(uint32_t size, uint8_t buf[]);
Function descriptions	write user data
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 32

Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write user data value */
uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                      0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};

ErrStatus flag = ERROR;

flag = efuse_user_data_write(32, (uint8_t *)buffer);
```

efuse_res_read

The description of efuse_res_read is shown as below:

Table 3-167. Function efuse_res_read

Function name	efuse_res_read
Function prototype	void efuse_res_read(uint32_t buf[]);
Function descriptions	read MCU reserved value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	MCU reserved value after system reset
Return value	
-	-

Example:

```
/* read MCU reserved value */
uint32_t buffer[3];

efuse_res_read(*buffer);
```

efuse_aes_key_read

The description of efuse_aes_key_read is shown as below:

Table 3-168. Function efuse_aes_key_read

Function name	efuse_aes_key_read
----------------------	--------------------

Function prototype	void efuse_aes_key_read(uint32_t buf[]);
Function descriptions	read AES key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	AES_KEY data after system reset
Return value	
-	-

Example:

```

/* read AES key */
uint32_t buffer[4];
efuse_aes_key_read(*buffer);

```

efuse_rotpk_key_read

The description of efuse_rotpk_key_read is shown as below:

Table 3-169. Function efuse_rotpk_key_read

Function name	efuse_rotpk_key_read
Function prototype	void efuse_rotpk_key_read(uint32_t size, uint32_t buf[]);
Function descriptions	read ROTPK key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	rotpk_key data after system reset
Return value	
-	-

Example:

```

/* read ROTPK key */
uint32_t buffer[8];
efuse_rotpk_key_read(*buffer);

```

efuse_puid_read

The description of efuse_puid_read is shown as below:

Table 3-170. Function efuse_puid_read

Function name	efuse_puid_read
Function prototype	void efuse_puid_read(uint32_t buf[]);
Function descriptions	read puid
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	puid data after system reset
Return value	
-	-

Example:

```
/* read puid */
uint32_t buffer[4];
efuse_puid_read(*buffer);
```

efuse_huk_key_read

The description of efuse_huk_key_read is shown as below:

Table 3-171. Function efuse_huk_key_read

Function name	efuse_huk_key_read
Function prototype	void efuse_huk_key_read(uint32_t buf[]);
Function descriptions	read huk key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	huk key data after system reset
Return value	
-	-

Example:

```
/* read huk key */
uint32_t buffer[4];
efuse_huk_key_read(*buffer);
```

efuse_user_data_read

The description of efuse_user_data_read is shown as below:

Table 3-172. Function efuse_user_data_read

Function name	efuse_user_data_read
Function prototype	void efuse_user_data_read(uint32_t buf[]);
Function descriptions	read user data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	user data after system reset
Return value	
-	-

Example:

```
/* read user data */
uint32_t buffer[8];
efuse_user_data_read(*buffer);
```

efuse_boot_address_get

The description of efuse_boot_address_get is shown as below:

Table 3-173. Function efuse_boot_address_get

Function name	efuse_boot_address_get
Function prototype	uint32_t efuse_boot_address_get(void);
Function descriptions	get boot address information
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	current boot address (0x0 – 0xFFFFFFFF)

Example:

```
/* get boot address */
uint32_t addr = 0;
```

```
addr = efuse_boot_address_get();
```

efuse_lock_config

The description of efuse_lock_config is shown as below:

Table 3-174. Function efuse_lock_config

Function name	efuse_lock_config
Function prototype	void efuse_lock_config(efuse_reg_lock_enum source);
Function descriptions	lock efuse filed
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies filed to lock, refer to Table 3-156. Enum efuse_reg_lock_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock EFUSE boot bits */
efuse_lock_config(EFUSE_BOOT_LOCK);
```

efuse_flag_get

The description of efuse_flag_get is shown as below:

Table 3-175. Function efuse_flag_get

Function name	efuse_flag_get
Function prototype	FlagStatus efuse_flag_get(efuse_flag_enum flag);
Function descriptions	check EFUSE flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	specifies to get a flag, refer to Table 3-151. Enum efuse_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EFUSE write operation complete flag status */
FlagStatus state = efuse_flag_get(EFUSE_PGIF);
```

efuse_flag_clear

The description of efuse_flag_clear is shown as below:

Table 3-176. Function efuse_flag_clear

Function name	efuse_flag_clear
Function prototype	void efuse_flag_clear(efuse_clear_flag_enum flag);
Function descriptions	clear EFUSE pending flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specifies to clear a flag, refer to Table 3-152. Enum efuse_clear_flag_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EFUSE write operation complete flag status */
```

```
efuse_flag_clear(EFUSE_PGIC);
```

efuse_interrupt_enable

The description of efuse_interrupt_enable is shown as below:

Table 3-177. Function efuse_interrupt_enable

Function name	efuse_interrupt_enable
Function prototype	void efuse_interrupt_enable(efuse_int_enum source);
Function descriptions	enable EFUSE interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies an interrupt to enable, refer to Table 3-153. Enum efuse_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EFUSE write operation complete interrupt */
```

```
efuse_interrupt_enable(EFUSE_INTEN_PG);
```

efuse_interrupt_disable

The description of efuse_interrupt_disable is shown as below:

Table 3-178. Function efuse_interrupt_disable

Function name	efuse_interrupt_disable
Function prototype	void efuse_interrupt_disable(efuse_int_enum source);
Function descriptions	disable EFUSE interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies an interrupt to disable, refer to Table 3-153. Enum efuse_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EFUSE write operation complete interrupt */
```

```
efuse_interrupt_disable(EFUSE_INTEN_PG);
```

efuse_interrupt_flag_get

The description of efuse_interrupt_flag_get is shown as below:

Table 3-179. Function efuse_interrupt_flag_get

Function name	efuse_interrupt_flag_get
Function prototype	FlagStatus efuse_interrupt_flag_get(efuse_int_flag_enum int_flag);
Function descriptions	check EFUSE interrupt flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	specifies to get a flag, refer to Table 3-154. Enum efuse_int_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EFUSE write operation complete interrupt flag status */
```

```
FlagStatus state = efuse_interrupt_flag_get(EFUSE_INT_PGIF);
```


efuse_interrupt_flag_clear

The description of efuse_interrupt_flag_clear is shown as below:

Table 3-180. Function efuse_interrupt_flag_clear

Function name	efuse_interrupt_flag_clear
Function prototype	void efuse_interrupt_flag_clear(efuse_clear_int_flag_enum int_flag);
Function descriptions	clear EFUSE pending interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	specifies to clear a flag, refer to Table 3-155. efuse_clear_int_flag_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EFUSE write operation complete interrupt flag status */
```

```
efuse_interrupt_flag_clear(EFUSE_INT_PGIC);
```

3.9. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 26 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.9.1](#), the EXTI firmware functions are introduced in chapter [3.9.2](#).

3.9.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

Table 3-181. EXTI Registers

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVEN	event enable register
EXTI_RTEN	rising edge trigger enable register
EXTI_FTEN	falling edge trigger enable register
EXTI_SWIEV	software interrupt event register
EXTI_PD	pending register

3.9.2. Descriptions of Peripheral functions

EXTI firmware functions are listed in the table shown as below:

Table 3-182. EXTI firmware function

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	disable the software interrupt event from EXTI line x
exti_flag_get	get EXTI line x interrupt pending flag
exti_flag_clear	clear EXTI line x interrupt pending flag
exti_interrupt_flag_get	get EXTI line x interrupt pending flag
exti_interrupt_flag_clear	clear EXTI line x interrupt pending flag

Enum exti_line_enum

Table 3-183. Enum exti_line_enum

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_19	EXTI line 19
EXTI_20	EXTI line 20

EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25

Enum exti_mode_enum

Table 3-184. Enum exti_mode_enum

Member name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

Enum exti_trig_type_enum

Table 3-185. Enum exti_trig_type_enum

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger
EXTI_TRIG_NONE	without rising edge or falling edge trigger

exti_deinit

The description of exti_deinit is shown as below:

Table 3-186. Function exti_deinit

Function name	exti_deinit
Function prototype	void exti_deinit(void);
Function descriptions	deinitialize the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the EXTI */
exti_deinit();
```

exti_init

The description of exti_init is shown as below:

Table 3-187. Function exti_init

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Input parameter{in}	
mode	EXTI mode, refer to Table 3-184. Enum exti_mode_enum
Input parameter{in}	
trig_type	trigger type, refer to Table 3-185. Enum exti_trig_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

exti_interrupt_enable

The description of exti_interrupt_enable is shown as below:

Table 3-188. Function exti_interrupt_enable

Function name	exti_interrupt_enable
Function prototype	void exti_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */
```

```
exti_interrupt_enable(EXTI_0);
```

exti_interrupt_disable

The description of exti_interrupt_disable is shown as below:

Table 3-189. Function exti_interrupt_disable

Function name	exti_interrupt_disable
Function prototype	void exti_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the interrupts from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```

exti_event_enable

The description of exti_event_enable is shown as below:

Table 3-190. Function exti_event_enable

Function name	exti_event_enable
Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

exti_event_disable

The description of exti_event_disable is shown as below:

Table 3-191. Function exti_event_disable

Function name	exti_event_disable
Function prototype	void exti_event_disable(exti_line_enum linex);
Function descriptions	disable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

exti_software_interrupt_enable

The description of exti_software_interrupt_enable is shown as below:

Table 3-192. Function exti_software_interrupt_enable

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);
```

exti_software_interrupt_disable

The description of exti_software_interrupt_disable is shown as below:

Table 3-193. Function exti_software_interrupt_disable

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

exti_flag_get

The description of exti_flag_get is shown as below:

Table 3-194. Function exti_flag_get

Function name	exti_flag_get
Function prototype	FlagStatus exti_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

exti_flag_clear

The description of exti_flag_clear is shown as below:

Table 3-195. Function exti_flag_clear

Function name	exti_flag_clear
---------------	-----------------

Function prototype	void exti_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 flag status */
exti_flag_clear(EXTI_0);
```

exti_interrupt_flag_get

The description of exti_interrupt_flag_get is shown as below:

Table 3-196. Function exti_interrupt_flag_get

Function name	exti_interrupt_flag_get
Function prototype	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

exti_interrupt_flag_clear

The description of exti_interrupt_flag_clear is shown as below:

Table 3-197. Function exti_interrupt_flag_clear

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag

Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-183. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

3.10. FMC

There is flash controller and option byte for GD32VW55x series. The FMC registers are listed in chapter [3.10.1](#). The FMC firmware functions are introduced in chapter [3.10.2](#).

3.10.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

Table 3-198. FMC Registers

Registers	Descriptions
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register
FMC_OBR	Option byte register
FMC_OBUSER	Option byte user register
FMC_OBWRP0	Option byte write protection area register 0
FMC_OBWRP1	Option byte write protection area register 1
FMC_NODEC0	NO-RTDEC region register 0
FMC_NODEC1	NO-RTDEC region register 1
FMC_NODEC2	NO-RTDEC region register 2
FMC_NODEC3	NO-RTDEC region register 3
FMC_OFRG	Offset region register
FMC_OFVR	Offset value register
FMC_PID0	Product ID0 register
FMC_PID1	Product ID1 register

Registers	Descriptions
FMC_RFT0	RF Trim register 0
FMC_RFT1	RF Trim register 1
FMC_WFTx (x=0..15)	WiFi Trim register x (x=0..15)

3.10.2. Descriptions of Peripheral functions

FMC firmware functions are listed in the table shown as below:

Table 3-199. FMC firmware function

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_word_program	FMC program a word at the corresponding address
fmc_continuous_program	FMC program continuously at the corresponding address
fmc_obr_function_enable	enable FMC OBR function
fmc_obr_function_disable	disable FMC OBR function
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option bytes modification start command
ob_reload	reload option bytes
ob_security_protection_config	configure the option bytes security protection
ob_user_write	program option bytes USER
ob_write_protection_config	configure write protection pages
fmc_no_rtdec_config	configure NO-RTDEC pages
fmc_offset_region_config	configure offset region
fmc_offset_value_config	configure offset value
fmc_wifi_trim_cal_get	get calibration value
fmc_wifi_trim_pa_get	get Power Amplifier bias tune value
fmc_wifi_trim_get	get wifi trim
fmc_pid_get	get product ID
ob_write_protection_get	get the value of option bytes write protection state, only applies to get the status of write/erase protection setting by Efuse
ob_user_get	get the value of option bytes USER
ob_security_protection_flag_get	get option bytes security protection state
fmc_flag_get	get FMC flag status
fmc_flag_clear	clear the FMC flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt

Function name	Function description
fmc_interrupt_flag_get	get FMC interrupt flag
fmc_interrupt_flag_clear	clear FMC interrupt flag

fmc_state_enum

Table 3-200. fmc_state_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_ERR	parameter error

fmc_unlock

The description of fmc_unlock is shown as below:

Table 3-201. Function fmc_unlock

Function name	fmc_unlock
Function prototype	void fmc_unlock (void);
Function descriptions	unlock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the main FMC operation */
```

```
fmc_unlock();
```

fmc_lock

The description of fmc_lock is shown as below:

Table 3-202. Function fmc_lock

Function name	fmc_lock
Function prototype	void fmc_lock(void);
Function descriptions	lock the main FMC operation
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the main FMC operation */
```

```
fmc_lock();
```

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-203. Function fmc_page_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	FMC erase page
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
page_address	target page address
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* erase page */
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
state = fmc_page_erase ( 0x08004000);
```

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

Table 3-204. Function fmc_mass_erase

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void);
Function descriptions	erase whole chip

Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```

/* erase the whole chip */

fmc_unlock();

fmc_state_enum state;

state = fmc_mass_erase();

```

fmc_word_program

The description of fmc_word_program is shown as below:

Table 3-205. Function fmc_word_program

Function name	fmc_word_program
Function prototype	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
Function descriptions	program a word at the corresponding address
Precondition	fmc_unlock, fmc_page_erase/fmc_mass_erase
The called functions	-
Input parameter{in}	
address	the address to program
data	word to program(0x00000000 - 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```

/* program a word at the corresponding address */

fmc_state_enum state;

fmc_unlock();

fmc_page_erase(0x08004000);

state = fmc_word_program (0x08004000, 0xaabbccdd);

```

fmc_continuous_program

The description of fmc_continuous_program is shown as below:

Table 3-206. Function fmc_continuous_program

Function name	fmc_continuous_program
Function prototype	fmc_state_enum fmc_continuous_program(uint32_t address, uint32_t data[], uint32_t size);
Function descriptions	FMC program data continuously at the corresponding address
Precondition	fmc_unlock, fmc_page_erase/fmc_mass_erase
The called functions	-
Input parameter{in}	
address	address to program, must be 4-byte aligned
Input parameter{in}	
data[]	data buffer to program
Input parameter{in}	
size	data buffer size in words
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* program data continuously at the corresponding address */

uint32_t data[10]= {0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567};

fmc_state_enum state;

fmc_unlock();

state = fmc_word_program (0x08004000, data[], 8);
```

fmc_obr_function_enable

The description of fmc_obr_function_enable is shown as below:

Table 3-207. Function fmc_obr_function_enable

Function name	fmc_obr_function_enable
Function prototype	void fmc_obr_function_enable(uint32_t obr_function);
Function descriptions	enable FMC OBR function
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
obr_function	the OBR function

<i>FMC_OBR_NWDG_HW</i>	watchdog status function
<i>FMC_OBR_NRST_STDBY</i>	no reset generated when entering Standby mode function
<i>FMC_OBR_NSRT_DPSLP</i>	no reset generated when entering Deepsleep mode function
<i>FMC_OBR_SRAM1_RST</i>	SRAM1 reset automatically function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable software watchdog function */
```

```
ob_unlock();
```

```
fmc_obr_function_enable(FMC_OBR_NWDG_HW);
```

fmc_obr_function_disable

The description of fmc_obr_function_disable is shown as below:

Table 3-208. Function fmc_obr_function_disable

Function name	fmc_obr_function_disable
Function prototype	void fmc_obr_function_disable(uint32_t obr_function);
Function descriptions	disable FMC OBR function
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
obr_function	the OBR function
<i>FMC_OBR_NWDG_HW</i>	watchdog status function
<i>FMC_OBR_NRST_STDBY</i>	no reset generated when entering Standby mode function
<i>FMC_OBR_NSRT_DPSLP</i>	no reset generated when entering Deepsleep mode function
<i>FMC_OBR_SRAM1_RST</i>	SRAM1 reset automatically function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hardware watchdog function */

ob_unlock();

fmc_obr_function_disable(FMC_OBR_NWDG_HW);
```

ob_unlock

The description of ob_unlock is shown as below:

Table 3-209. Function ob_unlock

Function name	ob_unlock
Function prototype	void ob_unlock(void);
Function descriptions	unlock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the option byte operation */

ob_unlock();
```

ob_lock

The description of ob_lock is shown as below:

Table 3-210. Function ob_lock

Function name	ob_lock
Function prototype	void ob_lock(void);
Function descriptions	lock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the option byte operation */
```

```
ob_lock();
```

ob_start

The description of ob_start is shown as below:

Table 3-211. Function ob_start

Function name	ob_start
Function prototype	void ob_start(void);
Function descriptions	send option bytes modification start command
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* program option bytes USER data */
```

```
ob_unlock( );
```

```
ob_user_write(0XFFFF);
```

```
ob_start();
```

ob_reload

The description of ob_reload is shown as below:

Table 3-212. Function ob_reload

Function name	ob_reload
Function prototype	void ob_reload(void);
Function descriptions	reload option bytes
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* program option bytes USER data */

ob_unlock();

ob_user_write(0XFFFF);

ob_start();

ob_reload();
```

ob_security_protection_config

The description of ob_security_protection_config is shown as below:

Table 3-213. Function ob_security_protection_config

Function name	ob_security_protection_config
Function prototype	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
Function descriptions	configure security protection
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_spc	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_SPC_P1</i>	security protection level 1
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* configure no security protection */

ob_unlock();

ob_security_protection_config (FMC_NSPC);

ob_start();
```

ob_user_write

The description of ob_user_write is shown as below:

Table 3-214. Function ob_user_write

Function name	ob_user_write
Function prototype	fmc_state_enum ob_user_write(uint16_t ob_user);

Function descriptions	program option bytes USER
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_user	option bytes user value
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* program option bytes USER data */
```

```
ob_unlock();
```

```
ob_user_write(0xFFFF);
```

```
ob_start();
```

ob_write_protection_config

The description of ob_write_protection_config is shown as below:

Table 3-215. Function ob_write_protection_config

Function name	ob_write_protection_config
Function prototype	fmc_state_enum ob_write_protection_config(uint32_t wrp_spage, uint32_t wrp_epage, uint32_t wrp_register_index);
Function descriptions	configure write protection pages
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
wrp_spage	start page of write protection area, 0~1023
Input parameter{in}	
wrp_epage	end page of write protection area, 0~1023
Input parameter{in}	
wrp_register_index	FMC_OBWRP _x register index
<i>OBWRP_INDEX0</i>	option byte write protection area register 0
<i>OBWRP_INDEX1</i>	option byte write protection area register 1
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* configure write protection pages */
```

```
ob_unlock();
```

```
ob_write_protection_config(WRP_REGION_SPAGE, WRP_REGION_EPAGE, OBWRP_IN
DEX0);
```

```
ob_start();
```

fmc_no_rtdec_config

The description of fmc_no_rtdec_config is shown as below:

Table 3-216. Function fmc_no_rtdec_config

Function name	fmc_no_rtdec_config
Function prototype	void fmc_no_rtdec_config(uint32_t nodec_spage, uint32_t nodec_epage, uint32_t nodec_register_index);
Function descriptions	configure NO-RTDEC pages
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
nodec_spage	start page of NO-RTDEC area, 0~0x03FF
Input parameter{in}	
nodec_epage	end page of NO-RTDEC area, 0~0x03FF
Input parameter{in}	
nodec_register_index	NO-RTDEC region register index
<i>NODEC_INDEX0</i>	NO-RTDEC region register 0
<i>NODEC_INDEX1</i>	NO-RTDEC region register 1
<i>NODEC_INDEX2</i>	NO-RTDEC region register 2
<i>NODEC_INDEX3</i>	NO-RTDEC region register 3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure NO-RTDEC pages */
```

```
ob_unlock();
```

```
fmc_no_rtdec_config(NODEC_REGION_SPAGE, NODEC_REGION_EPAGE, NODEC_IN
DEX0);
```

fmc_offset_region_config

The description of fmc_offset_region_config is shown as below:

Table 3-217. Function fmc_offset_region_config

Function name	fmc_offset_region_config
----------------------	--------------------------

Function prototype	void fmc_offset_region_config(uint32_t of_spage, uint32_t of_epage);
Function descriptions	configure offset region
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
of_spage	start page of offset region, 0~0x03FF
Input parameter{in}	
of_epage	end page of offset region, 0~0x03FF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure offset region */

ob_unlock();

fmc_offset_region_config(SOURCE_START_PAGE, SOURCE_END_PAGE);
```

fmc_offset_value_config

The description of fmc_offset_value_config is shown as below:

Table 3-218. Function fmc_offset_value_config

Function name	fmc_offset_value_config
Function prototype	void fmc_offset_value_config(uint32_t of_value);
Function descriptions	configure offset value
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
of_value	offset value, 0~0x1FFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure offset value */

ob_unlock();

fmc_offset_value_config(PAGE_OFFSET_VALUE);
```

fmc_wifi_trim_cal_get

The description of fmc_wifi_trim_cal_get is shown as below:

Table 3-219. Function fmc_wifi_trim_cal_get

Function name	fmc_wifi_trim_cal_get
Function prototype	void fmc_wifi_trim_cal_get(uint32_t *rft0_bletxcal, uint32_t *rft0_wftxcal, uint32_t *rft0_thecal, uint32_t *rft1_wfrxcal);
Function descriptions	get calibration value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
<i>rft0_bletxcal</i>	BLE transmit power calibration value
<i>rft0_wftxcal</i>	WIFI transmit power calibration value
<i>rft0_thecal</i>	thermal meter calibration value
<i>rft1_wfrxcal</i>	WIFI receive gain calibration value
Return value	
-	-

Example:

```
/* get calibration value */
uint32_t bletxcal, wftxcal, thecal, wfrxcal;
fmc_wifi_trim_cal_get(&bletxcal, &wftxcal, &thecal, &wfrxcal);
```

fmc_wifi_trim_pa_get

The description of fmc_wifi_trim_pa_get is shown as below:

Table 3-220. Function fmc_wifi_trim_pa_get

Function name	fmc_wifi_trim_cal_get
Function prototype	void fmc_wifi_trim_pa_get(uint32_t *rft0_pa_tune0, uint32_t *rft0_pa_tune1);
Function descriptions	get Power Amplifier bias tune value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
<i>rft0_pa_tune0</i>	Power Amplifier bias coarse tune value
<i>rft0_pa_tune1</i>	Power Amplifier bias fine tune value
Return value	

-	-
---	---

Example:

```
/* get Power Amplifier bias tune value */

uint32_t pa_tune0, pa_tune1;

fmc_wifi_trim_pa_get(&pa_tune0, &pa_tune1);
```

fmc_wifi_trim_get

The description of fmc_wifi_trim_get is shown as below:

Table 3-221. Function fmc_wifi_trim_get

Function name	fmc_wifi_trim_get
Function prototype	fmc_state_enum fmc_wifi_trim_get(uint32_t size, uint32_t buf[]);
Function descriptions	configure wifi trim
Precondition	-
The called functions	-
Input parameter{in}	
size	the size of data read from WIFI trim register, must be 16U
Output parameter{out}	
buf[]	the buffer of data read from WIFI trim register
Return value	
fmc_state_enum	state of FMC, refer to Table 3-200. fmc_state_enum

Example:

```
/* get the FMC WIFI trim register */

fmc_state_enum fmc_state = FMC_ERR;

uint32_t size = 16U;

uint32_t data[16];

fmc_state = fmc_wifi_trim_get(size, &data);
```

ob_write_protection_get

The description of ob_write_protection_get is shown as below:

Table 3-222. Function ob_write_protection_get

Function name	ob_write_protection_get
Function prototype	FlagStatus ob_write_protection_get(void);
Function descriptions	get write protection state, only applies to get the status of write/erase protection setting by EFUSE
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the FMC write protection status */
```

```
FlagStatus status;
```

```
status = ob_write_protection_get();
```

ob_user_get

The description of ob_user_get is shown as below:

Table 3-223. Function ob_user_get

Function name	ob_user_get
Function prototype	uint16_t ob_user_get(void);
Function descriptions	get the value of option bytes USER
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000~0xFFFF

Example:

```
/* get the value of option bytes USER */
```

```
uint16_t user_value;
```

```
user_value = ob_user_get();
```

ob_security_protection_flag_get

The description of ob_security_protection_flag_get is shown as below:

Table 3-224. Function ob_security_protection_flag_get

Function name	ob_security_protection_flag_get
Function prototype	FlagStatus ob_security_protection_flag_get(void);
Function descriptions	get the FMC option bytes security protection state
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check FMC option bytes security protection level 1 is set or not */
```

```
FlagStatus status;
```

```
status = ob_security_protection_flag_get();
```

fmc_flag_get

The description of fmc_flag_get is shown as below:

Table 3-225. Function fmc_flag_get

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

fmc_flag_clear

The description of fmc_flag_clear is shown as below:

Table 3-226. Function `fmc_flag_clear`

Function name	<code>fmc_flag_clear</code>
Function prototype	<code>void fmc_flag_clear(uint32_t flag);</code>
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
<code>FMC_FLAG_WPERR</code>	FMC erase/program protection error flag bit
<code>FMC_FLAG_END</code>	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_FLAG_END flag */
fmc_flag_clear(FMC_FLAG_END);
```

`fmc_interrupt_enable`

The description of `fmc_interrupt_enable` is shown as below:

Table 3-227. Function `fmc_interrupt_enable`

Function name	<code>fmc_interrupt_enable</code>
Function prototype	<code>void fmc_interrupt_enable (uint32_t interrupt);</code>
Function descriptions	enable FMC interrupt
Precondition	<code>fmc_unlock</code>
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<code>FMC_INT_END</code>	disable FMC end of program interrupt
<code>FMC_INT_ERR</code>	disable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FMC interrupt */
fmc_unlock();
fmc_interrupt_enable(FMC_INT_END);
```

fmc_interrupt_disable

The description of fmc_interrupt_disable is shown as below:

Table 3-228. Function fmc_interrupt_disable

Function name	fmc_interrupt_disable
Function prototype	void fmc_interrupt_disable(uint32_t interrupt);
Function descriptions	fmc_unlock
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	disable FMC end of program interrupt
<i>FMC_INT_ERR</i>	disable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable FMC interrupt */

fmc_unlock();

fmc_interrupt_disable(FMC_INT_END);
```

fmc_interrupt_flag_get

The description of fmc_interrupt_flag_get is shown as below:

Table 3-229. Function fmc_interrupt_flag_get

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(uint32_t flag);
Function descriptions	get FMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	the FMC interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check operation interrupt flag is set or not */
```

```
FlagStatus flag;
```

```
flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

fmc_interrupt_flag_clear

The description of fmc_interrupt_flag_clear is shown as below:

Table 3-230. Function fmc_interrupt_flag_clear

Function name	fmc_interrupt_flag_clear
Function prototype	void fmc_interrupt_flag_clear(uint32_t flag);
Function descriptions	clear FMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	the FMC interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear operation interrupt flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_END);
```

3.11. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.11.1](#), the FWDGT firmware functions are introduced in chapter [3.11.2](#).

3.11.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-231. FWDGT Registers

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register

3.11.2. Descriptions of Peripheral functions

FWDGT firmware functions are listed in the table shown as below:

Table 3-232. FWDGT firmware function

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the FWDGT counter
fwdgt_prescaler_value_config	configure the FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

fwdgt_write_enable

The description of fwdgt_write_enable is shown as below:

Table 3-233. Function fwdgt_write_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_enable();
```

fwdgt_write_disable

The description of fwdgt_write_disable is shown as below:

Table 3-234. Function fwdgt_write_disable

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_disable();

```

fwdgt_enable

The description of fwdgt_enable is shown as below:

Table 3-235. Function fwdgt_enable

Function name	fwdgt_enable
Function prototype	void fwdgt_enable(void);
Function descriptions	start the FWDGT counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* start the free watchdog timer counter */
fwdgt_enable();

```

fwdgt_prescaler_value_config

The description of fwdgt_prescaler_value_config is shown as below:

Table 3-236. Function fwdgt_prescaler_value_config

Function name	fwdgt_prescaler_value_config
Function prototype	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
Function descriptions	configure the FWDGT counter prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
prescaler_value	specify prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure the FWDGT counter prescaler value */
```

```
fwdgt_prescaler_value_config(FWDGT_PSC_DIV8);
```

fwdgt_reload_value_config

The description of fwdgt_reload_value_config is shown as below:

Table 3-237. Function fwdgt_reload_value_config

Function name	fwdgt_reload_value_config
Function prototype	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
Function descriptions	configure the FWDGT counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	specify reload value(0x0000 - 0x0FFF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure the FWDGT counter reload value */

fwdgt_reload_value_config(625);
```

fwdgt_counter_reload

The description of fwdgt_counter_reload is shown as below:

Table 3-238. Function fwdgt_counter_reload

Function name	fwdgt_counter_reload
Function prototype	void fwdgt_counter_reload(void);
Function descriptions	reload the counter of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reload FWDGT counter */

fwdgt_counter_reload();
```

fwdgt_config

The description of fwdgt_config is shown as below:

Table 3-239. Function fwdgt_config

Function name	fwdgt_config
Function prototype	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_value);
Function descriptions	configure counter reload value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	specify reload value(0x0000 - 0x0FFF)
Input parameter{in}	
prescaler_value	FWDGT prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32

<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

fwdgt_flag_get

The description of fwdgt_flag_get is shown as below:

Table 3-240. Function fwdgt_flag_get

Function name	fwdgt_flag_get
Function prototype	FlagStatus fwdgt_flag_get(uint16_t flag);
Function descriptions	get flag state of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;
```

```
status = fwdgt_flag_get(FWDGT_FLAG_PUD);
```

3.12. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.12.1](#), the GPIO firmware functions are introduced in chapter [3.12.2](#).

3.12.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-241. GPIO Registers

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPD	GPIO port output speed register
GPIOx_PUD	GPIO port pull-up/pull-down register
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_AFSEL0	GPIO alternate function selected register 0
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
GPIOx_TG	GPIO port bit toggle register

3.12.2. Descriptions of Peripheral functions

GPIO firmware functions are listed in the table shown as below:

Table 3-242. GPIO firmware function

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin bit
gpio_bit_toggle	toggle GPIO pin status
gpio_port_toggle	toggle GPIO port status

gpio_deinit

The description of gpio_deinit is shown as below:

Table 3-243. Function gpio_deinit

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

gpio_mode_set

The description of gpio_mode_set is shown as below:

Table 3-244. Function gpio_mode_set

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
mode	gpio pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	gpio pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors

<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i>	with pull-down resistor
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as input mode with pullup */
```

```
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

gpio_output_options_set

The description of gpio_output_options_set is shown as below:

Table 3-245. Function gpio_output_options_set

Function name	gpio_output_options_set
Function prototype	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Input parameter{in}	
otype	gpio pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode
Input parameter{in}	
speed	gpio pin output max speed
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_25MHZ</i>	output max speed 25MHz
<i>GPIO_OSPEED_MAX</i>	output max speed
Input parameter{in}	

pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as push pull mode */
```

```
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

gpio_bit_set

The description of gpio_bit_set is shown as below:

Table 3-246. Function gpio_bit_set

Function name	gpio_bit_set
Function prototype	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
Function descriptions	set GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

gpio_bit_reset

The description of gpio_bit_reset is shown as below:

Table 3-247. Function gpio_bit_reset

Function name	gpio_bit_reset
Function prototype	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
Function descriptions	reset GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

gpio_bit_write

The description of gpio_bit_write is shown as below:

Table 3-248. Function gpio_bit_write

Function name	gpio_bit_write
Function prototype	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
Function descriptions	write data to the specified GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Input parameter{in}	
bit_value	SET or RESET
RESET	clear the port pin
SET	set the port pin

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1 to PA0 */
```

```
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

gpio_port_write

The description of gpio_port_write is shown as below:

Table 3-249. Function gpio_port_write

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
data	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write(GPIOA, 0xA5A5);
```

gpio_input_bit_get

The description of gpio_input_bit_get is shown as below:

Table 3-250. Function gpio_input_bit_get

Function name	gpio_input_bit_get
Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-

Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);
```

gpio_input_port_get

The description of gpio_input_port_get is shown as below:

Table 3-251. Function gpio_input_port_get

Function name	gpio_input_port_get
Function prototype	uint16_t gpio_input_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO all pins input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

```
/* get input value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get(GPIOA);
```

gpio_output_bit_get

The description of gpio_output_bit_get is shown as below:

Table 3-252. Function gpio_output_bit_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get output status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

gpio_output_port_get

The description of gpio_output_port_get is shown as below:

Table 3-253. Function gpio_output_port_get

Function name	gpio_output_port_get
Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO all pins output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

```
/* get output value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

gpio_af_set

The description of gpio_af_set is shown as below:

Table 3-254. Function gpio_af_set

Function name	gpio_af_set
Function prototype	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
Function descriptions	set GPIO alternate function
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Input parameter{in}	
alt_func_num	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	USART0, UART1, I2C1, CK_OUT0, SPI, JTAG, CK_OUT1, RTC_REF, IFRP_OUT
<i>GPIO_AF_1</i>	TIMER0, TIMER1
<i>GPIO_AF_2</i>	USART0, TIMER0, TIMER2, SPI
<i>GPIO_AF_3</i>	QSPI, USART1, TIMER0, TIMER1, TIMER2
<i>GPIO_AF_4</i>	UART1, SPI, I2C0, QSPI, I2C1
<i>GPIO_AF_5</i>	SPI, I2C0
<i>GPIO_AF_6</i>	TIMER0, I2C0, SPI, I2C1
<i>GPIO_AF_7</i>	USART0, UART1, TIMER16, SPI
<i>GPIO_AF_8</i>	USART0, UART1, UART2, TIMER0, TIMER2, TIMER15
<i>GPIO_AF_9</i>	RTC, TIMER0, TIMER1, TIMER16, IFRP_OUT
<i>GPIO_AF_10</i>	UART2, TIMER2, TIMER16
<i>GPIO_AF_11</i>	TIMER0, TIMER15
<i>GPIO_AF_12</i>	-
<i>GPIO_AF_13</i>	-
<i>GPIO_AF_14</i>	-
<i>GPIO_AF_15</i>	EVENTOUT
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*set PA0 alternate function 0 */
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

gpio_pin_lock

The description of gpio_pin_lock is shown as below:

Table 3-255. Function gpio_pin_lock

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
Function descriptions	lock GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0 */
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

gpio_bit_toggle

The description of gpio_bit_toggle is shown as below:

Table 3-256. Function gpio_bit_toggle

Function name	gpio_bit_toggle
Function prototype	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
Function descriptions	toggle GPIO pin status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)

Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	GPIO_PIN_ALL
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle PA0 */
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

gpio_port_toggle

The description of gpio_port_toggle is shown as below:

Table 3-257. Function gpio_port_toggle

Function name	gpio_port_toggle
Function prototype	void gpio_port_toggle(uint32_t gpio_periph);
Function descriptions	toggle GPIO port status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA */
gpio_port_toggle(GPIOA);
```

3.13. HAU

The HASH Acceleration Unit supports acceleration of SHA-1, SHA-224, SHA-256, MD5 algorithm and the HMAC (keyed-hash message authentication code) algorithm. The HAU registers are listed in chapter [3.13.1](#). the HAU firmware functions are introduced in chapter [3.13.2](#).

3.13.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

Table 3-258. HAU Registers

Registers	Descriptions
HAU_CTL	control register
HAU_DI	data input register
HAU_CFG	configuration register
HAU_DO0	data output register 0
HAU_DO1	data output register 1
HAU_DO2	data output register 2
HAU_DO3	data output register 3
HAU_DO4	data output register 4
HAU_DO5	data output register 5
HAU_DO6	data output register 6
HAU_DO7	data output register 7
HAU_INTEN	interrupt enable register
HAU_STAT	status and interrupt flag register
HAU_CTXSx (x=0..53)	context switch register

3.13.2. Descriptions of Peripheral functions

HAU firmware functions are listed in the table shown as below:

Table 3-259. HAU firmware function

Function name	Function description
hau_deinit	reset the HAU peripheral
hau_init	initialize the HAU peripheral parameters
hau_init_struct_para_init	initialize the structure hau_initpara
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO
hau_infifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_context_struct_para_init	initialize the struct context
hau_context_save	save the HAU peripheral context

Function name	Function description
hau_context_restore	restore the HAU peripheral context
hau_hash_sha_1	calculate digest using SHA1 in HASH mode
hau_hmac_sha_1	calculate digest using SHA1 in HMAC mode
hau_hash_sha_224	calculate digest using SHA224 in HASH mode
hau_hmac_sha_224	calculate digest using SHA224 in HMAC mode
hau_hash_sha_256	calculate digest using SHA256 in HASH mode
hau_hmac_sha_256	calculate digest using SHA256 in HMAC mode
hau_hash_md5	calculate digest using MD5 in HASH mode
hau_hmac_md5	calculate digest using MD5 in HMAC mode
hau_flag_get	get the HAU flag status
hau_flag_clear	clear the HAU flag status
hau_interrupt_enable	enable the HAU interrupts
hau_interrupt_disable	disable the HAU interrupts
hau_interrupt_flag_get	get the HAU interrupt flag status
hau_interrupt_flag_clear	clear the HAU interrupt flag status

Structure hau_init_parameter_struct

Table 3-260. Structure hau_init_parameter_struct

Member name	Function description
algo	algorithm selection: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU mode selection: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	data type mode: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	key length mode: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

Structure hau_digest_parameter_struct

Table 3-261. Structure hau_digest_parameter_struct

Member name	Function description
out[8]	message digest result 0-7

Structure hau_context_parameter_struct

Table 3-262. Structure hau_context_parameter_struct

Member name	Function description
hau_ctl_bak	backup of HAU_CTL register
hau_cfg_bak	backup of HAU_CFG register
hau_inten_bak	backup of HAU_INTEN register
hau_ctxs_bak[54]	backup of HAU_CTXSx registers

hau_deinit

The description of hau_deinit is shown as below:

Table 3-263. Function hau_deinit

Function name	hau_deinit
Function prototype	void hau_deinit(void);
Function descriptions	reset the HAU peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the HAU peripheral */
hau_deinit();
```

hau_init

The description of hau_init is shown as below:

Table 3-264. Function hau_init

Function name	hau_init
Function prototype	void hau_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the HAU peripheral parameters
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	the HAU peripheral parameters, refer to structure Table 3-260. Structure hau_init_parameter_struct
Return value	
-	-

Example:

```
/* initialize the HAU peripheral parameters */
hau_init_parameter_struct hau_initpara;
```

...

```

hau_init_struct_para_init(&hau_initpara);

hau_initpara.algo = algo;

hau_initpara.mode = HAU_MODE_HMAC;

hau_initpara.datatype = HAU_SWAPPING_8BIT;

if(key_len > 64U){

    hau_initpara.keytype = HAU_KEY_LONGGER_64;

}else{

    hau_initpara.keytype = HAU_KEY_SHORTER_64;

}

hau_init(&hau_initpara);

```

hau_init_struct_para_init

The description of hau_init_struct_para_init is shown as below:

Table 3-265. Function hau_init_struct_para_init

Function name	hau_init_struct_para_init
Function prototype	void hau_init_struct_para_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the struct hau_initpara
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	the HAU peripheral parameters, refer to structure Table 3-260. Structure hau_init_parameter_struct
Return value	
-	-

Example:

```

/* initialize the HAU peripheral parameters */

hau_init_parameter_struct hau_initpara;

hau_init_struct_para_init(&hau_initpara);

```

hau_reset

The description of hau_reset is shown as below:

Table 3-266. Function hau_reset

Function name	hau_reset
Function prototype	void hau_reset(void);
Function descriptions	reset the HAU processor core
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the HAU processor core */
hau_reset();
```

hau_last_word_validbits_num_config

The description of hau_last_word_validbits_num_config is shown as below:

Table 3-267. Function hau_last_word_validbits_num_config

Function name	hau_last_word_validbits_num_config
Function prototype	void hau_last_word_validbits_num_config(uint32_t valid_num);
Function descriptions	configure the number of valid bits in last word of the message
Precondition	-
The called functions	-
Input parameter{in}	
valid_num	number of valid bits in last word of the message(0x00 – 0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of valid bits in last word of the message */
hau_last_word_validbits_num_config(0x10);
```

hau_data_write

The description of hau_data_write is shown as below:

Table 3-268. Function hau_data_write

Function name	hau_data_write
Function prototype	void hau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write (0x0 – 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data to the IN FIFO */
```

```
hau_data_write(0x10);
```

hau_infifo_words_num_get

The description of hau_infifo_words_num_get is shown as below:

Table 3-269. Function hau_infifo_words_num_get

Function name	hau_infifo_words_num_get
Function prototype	uint32_t hau_infifo_words_num_get(void);
Function descriptions	return the number of words already written into the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num;
```

```
num = hau_infifo_words_num_get();
```

hau_digest_read

The description of hau_digest_read is shown as below:

Table 3-270. Function hau_digest_read

Function name	hau_digest_read
Function prototype	void hau_digest_read(hau_digest_parameter_struct* digestpara);
Function descriptions	read the message digest result
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
digestpara	the message digest result, refer to structure Table 3-261. Structure hau_digest_parameter_struct
Return value	
-	-

Example:

```
/* read the message digest result */
hau_digest_parameter_struct digestpara;
hau_digest_read(&digestpara);
```

hau_digest_calculation_enable

The description of hau_digest_calculation_enable is shown as below:

Table 3-271. Function hau_digest_calculation_enable

Function name	hau_digest_calculation_enable
Function prototype	void hau_digest_calculation_enable(void);
Function descriptions	enable digest calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable digest calculation */
hau_digest_calculation_enable();
```

hau_multiple_single_dma_config

The description of hau_multiple_single_dma_config is shown as below:

Table 3-272. Function hau_multiple_single_dma_config

Function name	hau_multiple_single_dma_config
Function prototype	void hau_multiple_single_dma_config(uint32_t multi_single);
Function descriptions	configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not
Precondition	-
The called functions	-
Input parameter{in}	
multi_single	Multiple or single
<i>SINGLE_DMA_AUTO_DIGEST</i>	message padding and message digest calculation at the end of a DMA transfer
<i>MULTIPLE_DMA_NO_DIGEST</i>	multiple DMA transfers needed and CALEN bit is not automatically set at the end of a DMA transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

hau_dma_enable

The description of hau_dma_enable is shown as below:

Table 3-273. Function hau_dma_enable

Function name	hau_dma_enable
Function prototype	void hau_dma_enable(void);
Function descriptions	enable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HAU DMA interface */
```

```
hau_dma_enable();
```

hau_dma_disable

The description of hau_dma_disable is shown as below:

Table 3-274. Function hau_dma_disable

Function name	hau_dma_disable
Function prototype	void hau_dma_disable(void);
Function descriptions	disable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HAU DMA interface */
hau_dma_disable();
```

hau_context_struct_para_init

The description of hau_context_struct_para_init is shown as below:

Table 3-275. Function hau_context_struct_para_init

Function name	hau_context_struct_para_init
Function prototype	void hau_context_struct_para_init(hau_context_parameter_struct* context);
Function descriptions	initialize the struct context
Precondition	-
The called functions	-
Input parameter{in}	
context	the context, refer to structure Table 3-262. Structure hau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct context */
hau_context_parameter_struct context_para;
hau_context_struct_para_init(&context_para);
```

hau_context_save

The description of hau_context_save is shown as below:

Table 3-276. Function hau_context_save

Function name	hau_context_save
Function prototype	void hau_context_save(hau_context_parameter_struct* context_save);
Function descriptions	save the HAU peripheral context
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
context_save	the HAU peripheral context, refer to structure Table 3-262. Structure hau_context_parameter_struct
Return value	
-	-

Example:

```
hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

/* save HAU context */

hau_context_save(&context_para);
```

hau_context_restore

The description of hau_context_restore is shown as below:

Table 3-277. Function hau_context_restore

Function name	hau_context_restore
Function prototype	void hau_context_restore(hau_context_parameter_struct* context_restore);
Function descriptions	restore the HAU peripheral context
Precondition	-
The called functions	-
Input parameter{in}	
context_restore	the HAU peripheral context, refer to structure Table 3-262. Structure hau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
hau_context_parameter_struct context_para;
```

```
hau_context_struct_para_init(&context_para);
```

```
hau_context_save(&context_para);
```

```
.....
```

```
/* restore HAU context */
```

```
hau_context_restore(&context_para);
```

hau_hash_sha_1

The description of hau_hash_sha_1 is shown as below:

Table 3-278. Function hau_hash_sha_1

Function name	hau_hash_sha_1
Function prototype	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]);
Function descriptions	calculate digest using SHA1 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA1 in HASH mode */
```

```
static uint8_t output[20];
```

```
ErrStatus status;
```

```
status = hau_hash_sha_1(&input, 0x10, output[0]);
```

hau_hmac_sha_1

The description of hau_hmac_sha_1 is shown as below:

Table 3-279. Function hau_hmac_sha_1

Function name	hau_hmac_sha_1
Function prototype	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);

Function descriptions	calculate digest using SHA1 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* calculate digest using SHA1 in HMAC mode */

static uint8_t output[20];

ErrStatus status;

status = hau_hmac_sha_1(&key, 0x10, &input, 0x10, output[0]);

```

hau_hash_sha_224

The description of hau_hash_sha_224 is shown as below:

Table 3-280. Function hau_hash_sha_224

Function name	hau_hash_sha_224
Function prototype	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]);
Function descriptions	calculate digest using SHA224 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:


```

/* calculate digest using SHA224 in HASH mode */

static uint8_t output[20];

ErrStatus status;

status = hau_hash_sha_224 (&input, 0x10, output[0]);

```

hau_hmac_sha_224

The description of hau_hmac_sha_224 is shown as below:

Table 3-281. Function hau_hmac_sha_224

Function name	hau_hmac_sha_224
Function prototype	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]);
Function descriptions	calculate digest using SHA224 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* calculate digest using SHA224 in HMAC mode */

static uint8_t output[20];

ErrStatus status;

status = hau_hmac_sha_224(&key, 0x10, &input, 0x10, output[0]);

```

hau_hash_sha_256

The description of hau_hash_sha_256 is shown as below:

Table 3-282. Function hau_hash_sha_256

Function name	hau_hash_sha_256
Function prototype	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t

	output[32]);
Function descriptions	calculate digest using SHA256 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HASH mode */

static uint8_t output[20];

ErrStatus status;

status = hau_hash_sha_256(&input, 0x10, output[0]);
```

hau_hmac_sha_256

The description of hau_hmac_sha_256 is shown as below:

Table 3-283. Function hau_hmac_sha_256

Function name	hau_hmac_sha_256
Function prototype	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]);
Function descriptions	calculate digest using SHA256 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HMAC mode */

static uint8_t output[20];

ErrStatus status;

status = hau_hmac_sha_256 (&key, 0x10, &input, 0x10, output[0]);
```

hau_hash_md5

The description of hau_hash_md5 is shown as below:

Table 3-284. Function hau_hash_md5

Function name	hau_hash_md5
Function prototype	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]);
Function descriptions	calculate digest using MD5 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HASH mode */

static uint8_t output[16];

ErrStatus status;

status = hau_hash_md5 (&input, 0x10, output[0]);
```

hau_hmac_md5

The description of hau_hmac_md5 is shown as below:

Table 3-285. Function hau_hmac_md5

Function name	hau_hmac_md5
Function prototype	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]);
Function descriptions	calculate digest using MD5 in HMAC mode

Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HMAC mode */
```

```
static uint8_t output[16];
```

```
ErrStatus status;
```

```
status = hau_hmac_md5 (&key, 0x10, &input, 0x10, output[0]);
```

hau_flag_get

The description of hau_flag_get is shown as below:

Table 3-286. Function hau_flag_get

Function name	hau_flag_get
Function prototype	FlagStatus hau_flag_get(uint32_t flag);
Function descriptions	get the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<i>HAU_FLAG_DMA</i>	DMA is enabled (DMAE =1) or a transfer is processing
<i>HAU_FLAG_BUSY</i>	data block is in process
<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	the input FIFO is not empty
Output parameter{out}	

-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get (HAU_FLAG_DMA);
```

hau_flag_clear

The description of hau_flag_clear is shown as below:

Table 3-287. Function hau_flag_clear

Function name	hau_flag_clear
Function prototype	void hau_flag_clear(uint32_t flag);
Function descriptions	clear the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the HAU flag status */
```

```
hau_flag_clear (HAU_FLAG_DATA_INPUT);
```

hau_interrupt_enable

The description of hau_interrupt_enable is shown as below:

Table 3-288. Function hau_interrupt_enable

Function name	hau_interrupt_enable
Function prototype	void hau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable the HAU interrupts
Precondition	-

The called functions	-
Input parameter{in}	
interrupt	HAU flag status
<i>HAU_INT_DATA_INPUT_T</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATION_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

hau_interrupt_disable

The description of `hau_interrupt_disable` is shown as below:

Table 3-289. Function `hau_interrupt_disable`

Function name	<code>hau_interrupt_disable</code>
Function prototype	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
Function descriptions	disable the HAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	HAU flag status
<i>HAU_INT_DATA_INPUT_T</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATION_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hau interrupt */
```

```
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

hau_interrupt_flag_get

The description of hau_interrupt_flag_get is shown as below:

Table 3-290. Function hau_interrupt_flag_get

Function name	hau_interrupt_flag_get
Function prototype	FlagStatus hau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the HAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU interrupt flag status */
```

```
FlagStatus status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

hau_interrupt_flag_clear

The description of hau_interrupt_flag_clear is shown as below:

Table 3-291. Function hau_interrupt_flag_clear

Function name	hau_interrupt_flag_clear
Function prototype	void hau_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the HAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* clear the HAU interrupt flag status */
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

3.14. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.14.1](#), the I2C firmware functions are introduced in chapter [3.14.2](#).

3.14.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-292. I2C Registers

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_TIMING	Timing register
I2C_TIMEOUT	Timeout register
I2C_STAT	Status register
I2C_STATC	I2C status clear register
I2C_PEC	PEC register
I2C_RDATA	Receive data register
I2C_TDATA	Transmit data register
I2C_CTL2	Control register 2

3.14.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

Table 3-293. I2C firmware function

Function name	Function description
i2c_deinit	reset I2C
i2c_timing_config	configure the timing parameters
i2c_digital_noise_filter_config	configure digital noise filter
i2c_analog_noise_filter_enable	enable analog noise filter
i2c_analog_noise_filter_disable	disable analog noise filter

Function name	Function description
i2c_master_clock_config	configure the SCL high and low period of clock in master mode
i2c_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_address10_enable	enable 10-bit addressing mode in master mode
i2c_address10_disable	disable 10-bit addressing mode in master mode
i2c_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_slave_response_to_gcall_enable	enable the response to a general call
i2c_slave_response_to_gcall_disable	disable the response to a general call
i2c_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_address_config	configure i2c slave address
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_address_disable	disable i2c address in slave mode
i2c_second_address_config	configure i2c second slave address
i2c_second_address_disable	disable i2c second address in slave mode
i2c_receved_address_get	get received match address in slave mode
i2c_slave_byte_control_enable	enable slave byte control
i2c_slave_byte_control_disable	disable slave byte control
i2c_nack_enable	generate a NACK in slave mode
i2c_wakeup_from_deepsleep_enable	enable wakeup from Deep-sleep mode
i2c_wakeup_from_deepsleep_disable	disable wakeup from Deep-sleep mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data
i2c_data_receive	I2C receive data
i2c_reload_enable	enable I2C reload mode
i2c_reload_disable	disable I2C reload mode
i2c_transfer_byte_number_config	configure number of bytes to be transferred
i2c_dma_enable	enable I2C DMA for transmission or reception
i2c_dma_disable	disable I2C DMA for transmission or reception

Function name	Function description
i2c_pec_transfer	I2C transfers PEC value
i2c_pec_enable	enable I2C PEC calculation
i2c_pec_disable	disable I2C PEC calculation
i2c_pec_value_get	get packet error checking value
i2c_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert
i2c_smbus_default_addr_enable	enable SMBus device default address
i2c_smbus_default_addr_disable	disable SMBus device default address
i2c_smbus_host_addr_enable	enable SMBus Host address
i2c_smbus_host_addr_disable	disable SMBus Host address
i2c_extented_clock_timeout_enable	enable extended clock timeout detection
i2c_extented_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

Enum i2c_interrupt_flag_enum

Table 3-294. i2c_interrupt_flag_enum

Member name	Function description
I2C_INT_FLAG_TI	transmit interrupt flag
I2C_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag
I2C_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C_INT_FLAG_NACK	not acknowledge interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C_INT_FLAG_BERR	bus error interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C_INT_FLAG_OUERR	overrun/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error interrupt flag
I2C_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert interrupt flag

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-295. Function i2c_deinit

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

i2c_timing_config

The description of i2c_timing_config is shown as below:

Table 3-296. Function i2c_timing_config

Function name	i2c_timing_config
Function prototype	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
Function descriptions	configure the timing parameters
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
psc	0-0xf, timing prescaler
Input parameter{in}	
scl_dely	0-0xf, data setup time
Input parameter{in}	
sda_dely	0-0xf, data hold time
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

i2c_digital_noise_filter_config

The description of i2c_digital_noise_filter_config is shown as below:

Table 3-297. Function i2c_digital_noise_filter_config

Function name	i2c_digital_noise_filter_config
Function prototype	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
Function descriptions	configure digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
filter_length	filter_length
FILTER_DISABLE	digital filter is disabled
FILTER_LENGTH_1	digital filter is enabled and filter spikes with a length of up to 1 tI2CCLK
FILTER_LENGTH_2	digital filter is enabled and filter spikes with a length of up to 2 tI2CCLK
FILTER_LENGTH_3	digital filter is enabled and filter spikes with a length of up to 3 tI2CCLK
FILTER_LENGTH_4	digital filter is enabled and filter spikes with a length of up to 4 tI2CCLK
FILTER_LENGTH_5	digital filter is enabled and filter spikes with a length of up to 5 tI2CCLK
FILTER_LENGTH_6	digital filter is enabled and filter spikes with a length of up to 6 tI2CCLK
FILTER_LENGTH_7	digital filter is enabled and filter spikes with a length of up to 7 tI2CCLK
FILTER_LENGTH_8	digital filter is enabled and filter spikes with a length of up to 8 tI2CCLK
FILTER_LENGTH_9	digital filter is enabled and filter spikes with a length of up to 9 tI2CCLK
FILTER_LENGTH_10	digital filter is enabled and filter spikes with a length of up to 10 tI2CCLK
FILTER_LENGTH_11	digital filter is enabled and filter spikes with a length of up to 11 tI2CCLK
FILTER_LENGTH_12	digital filter is enabled and filter spikes with a length of up to 12 tI2CCLK
FILTER_LENGTH_13	digital filter is enabled and filter spikes with a length of up to 13 tI2CCLK
FILTER_LENGTH_14	digital filter is enabled and filter spikes with a length of up to 14 tI2CCLK
FILTER_LENGTH_15	digital filter is enabled and filter spikes with a length of up to 15 tI2CCLK
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

i2c_analog_noise_filter_enable

The description of i2c_analog_noise_filter_enable is shown as below:

Table 3-298. Function i2c_analog_noise_filter_enable

Function name	i2c_analog_noise_filter_enable
Function prototype	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
Function descriptions	enable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

i2c_analog_noise_filter_disable

The description of i2c_analog_noise_filter_disable is shown as below:

Table 3-299. Function i2c_analog_noise_filter_disable

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	disable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable analog noise filter */

i2c_analog_noise_filter_disable(I2C0);
```

i2c_master_clock_config

The description of i2c_master_clock_config is shown as below:

Table 3-300. Function i2c_master_clock_config

Function name	i2c_master_clock_config
Function prototype	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
Function descriptions	configure the SCL high and low period of clock in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
sclh	0-0xff, SCL high period
Input parameter{in}	
scll	0-0xff, SCL low period
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */

i2c_master_clock_config (I2C0, 0x0f, 0x0f);
```

i2c_master_addressing

The description of i2c_master_addressing is shown as below:

Table 3-301. Function i2c_master_addressing

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
Function descriptions	configure i2c slave addresss and transfer direction in master mode
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
address	0-0x3FF except reserved address, I2C slave address to be sent
Input parameter{in}	
trans_direction	I2C transfer direction in master mode
<i>I2C_MASTER_TRANSMIT</i>	master transmit
<i>I2C_MASTER_RECEIVE</i>	master receive
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing (I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

i2c_address10_header_enable

The description of i2c_address10_header_enable is shown as below:

Table 3-302. Function i2c_address10_header_enable

Function name	i2c_address10_header_enable
Function prototype	void i2c_address10_header_enable(uint32_t i2c_periph);
Function descriptions	10-bit address header executes read direction only in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

i2c_address10_header_disable

The description of i2c_address10_header_disable is shown as below:

Table 3-303. Function i2c_address10_header_disable

Function name	i2c_address10_header_disable
Function prototype	void i2c_address10_header_disable(uint32_t i2c_periph);
Function descriptions	10-bit address header executes complete sequence in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

i2c_address10_enable

The description of i2c_address10_enable is shown as below:

Table 3-304. Function i2c_address10_enable

Function name	i2c_address10_enable
Function prototype	void i2c_address10_enable(uint32_t i2c_periph);
Function descriptions	enable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```


i2c_address10_disable

The description of i2c_address10_disable is shown as below:

Table 3-305. Function i2c_address10_disable

Function name	i2c_address10_disable
Function prototype	void i2c_address10_disable(uint32_t i2c_periph);
Function descriptions	disable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

i2c_automatic_end_enable

The description of i2c_automatic_end_enable is shown as below:

Table 3-306. Function i2c_automatic_end_enable

Function name	i2c_automatic_end_enable
Function prototype	void i2c_automatic_end_enable(uint32_t i2c_periph);
Function descriptions	enable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

i2c_automatic_end_disable

The description of i2c_automatic_end_disable is shown as below:

Table 3-307. Function i2c_automatic_end_disable

Function name	i2c_automatic_end_disable
Function prototype	void i2c_automatic_end_disable(uint32_t i2c_periph);
Function descriptions	disable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

i2c_slave_response_to_gcall_enable

The description of i2c_slave_response_to_gcall_enable is shown as below:

Table 3-308. Function i2c_slave_response_to_gcall_enable

Function name	i2c_slave_response_to_gcall_enable
Function prototype	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
Function descriptions	enable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

i2c_slave_response_to_gcall_disable

The description of i2c_slave_response_to_gcall_disable is shown as below:

Table 3-309. Function i2c_slave_response_to_gcall_disable

Function name	i2c_slave_response_to_gcall_disable
Function prototype	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
Function descriptions	disable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable(I2C0);
```

i2c_stretch_scl_low_enable

The description of i2c_stretch_scl_low_enable is shown as below:

Table 3-310. Function i2c_stretch_scl_low_enable

Function name	i2c_stretch_scl_low_enable
Function prototype	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
Function descriptions	enable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_enable(I2C0);
```

i2c_stretch_scl_low_disable

The description of i2c_stretch_scl_low_disable is shown as below:

Table 3-311. Function i2c_stretch_scl_low_disable

Function name	i2c_stretch_scl_low_disable
Function prototype	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
Function descriptions	disable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

i2c_address_config

The description of i2c_address_config is shown as below:

Table 3-312. Function i2c_address_config

Function name	i2c_address_config
Function prototype	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
Function descriptions	configure i2c slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_format	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_address_config (I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

i2c_address_bit_compare_config

The description of i2c_address_bit_compare_config is shown as below:

Table 3-313. Function i2c_address_bit_compare_config

Function name	i2c_address_bit_compare_config
Function prototype	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
compare_bits	the bits need to compare
<i>ADDRESS_BIT1_COMPARE</i>	address bit1 needs compare
<i>ADDRESS_BIT2_COMPARE</i>	address bit2 needs compare
<i>ADDRESS_BIT3_COMPARE</i>	address bit3 needs compare
<i>ADDRESS_BIT4_COMPARE</i>	address bit4 needs compare
<i>ADDRESS_BIT5_COMPARE</i>	address bit5 needs compare
<i>ADDRESS_BIT6_COMPARE</i>	address bit6 needs compare
<i>ADDRESS_BIT7_COMPARE</i>	address bit7 needs compare
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* bit 1 of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

i2c_address_disable

The description of i2c_address_disable is shown as below:

Table 3-314. Function i2c_address_disable

Function name	i2c_address_disable
Function prototype	void i2c_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```

i2c_second_address_config

The description of i2c_second_address_config is shown as below:

Table 3-315. Function i2c_second_address_config

Function name	i2c_second_address_config
Function prototype	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
Function descriptions	configure i2c second slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	

address	I2C address
Input parameter{in}	
addr_mask	the bits not need to compare
ADDRESS2_NO_MASK	no mask, all the bits must be compared
ADDRESS2_MASK_BIT1	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
ADDRESS2_MASK_ALL	all the ADDRESS2[7:1] bits are masked
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_second_address_config (I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

i2c_second_address_disable

The description of i2c_second_address_disable is shown as below:

Table 3-316. Function i2c_second_address_disable

Function name	i2c_second_address_disable
Function prototype	void i2c_second_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c second address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

i2c_receved_address_get

The description of i2c_receved_address_get is shown as below:

Table 3-317. Function i2c_receved_address_get

Function name	i2c_receved_address_get
Function prototype	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
Function descriptions	get received match address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	0x00..0x7F

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receved_address_get(I2C0);
```

i2c_slave_byte_control_enable

The description of i2c_slave_byte_control_enable is shown as below:

Table 3-318. Function i2c_slave_byte_control_enable

Function name	i2c_slave_byte_control_enable
Function prototype	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
Function descriptions	enable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable slave byte control */
i2c_slave_byte_control_enable(I2C0);
```

i2c_slave_byte_control_disable

The description of i2c_slave_byte_control_disable is shown as below:

Table 3-319. Function i2c_slave_byte_control_disable

Function name	i2c_slave_byte_control_disable
Function prototype	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
Function descriptions	disable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable slave byte control */
i2c_slave_byte_control_disable(I2C0);
```

i2c_nack_enable

The description of i2c_nack_enable is shown as below:

Table 3-320. Function i2c_nack_enable

Function name	i2c_nack_enable
Function prototype	void i2c_nack_enable(uint32_t i2c_periph);
Function descriptions	generate a NACK in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

i2c_wakeup_from_deepsleep_enable

The description of i2c_wakeup_from_deepsleep_enable is shown as below:

Table 3-321. Function i2c_wakeup_from_deepsleep_enable

Function name	i2c_wakeup_from_deepsleep_enable
Function prototype	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
Function descriptions	enable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable(I2C0);
```

i2c_wakeup_from_deepsleep_disable

The description of i2c_wakeup_from_deepsleep_disable is shown as below:

Table 3-322. Function i2c_wakeup_from_deepsleep_disable

Function name	i2c_wakeup_from_deepsleep_disable
Function prototype	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
Function descriptions	disable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */
i2c_wakeup_from_deepsleep_disable(I2C0);
```

i2c_enable

The description of i2c_enable is shown as below:

Table 3-323. Function i2c_enable

Function name	i2c_enable
Function prototype	void i2c_enable(uint32_t i2c_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */
i2c_enable(I2C0);
```

i2c_disable

The description of i2c_disable is shown as below:

Table 3-324. Function i2c_disable

Function name	i2c_disable
Function prototype	void i2c_disable(uint32_t i2c_periph);
Function descriptions	disable I2C
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 */
i2c_disable(I2C0);
```

i2c_start_on_bus

The description of i2c_start_on_bus is shown as below:

Table 3-325. Function i2c_start_on_bus

Function name	i2c_start_on_bus
Function prototype	void i2c_start_on_bus(uint32_t i2c_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus(I2C0);
```

i2c_stop_on_bus

The description of i2c_stop_on_bus is shown as below:

Table 3-326. Function i2c_stop_on_bus

Function name	i2c_stop_on_bus
Function prototype	void i2c_stop_on_bus(uint32_t i2c_periph);
Function descriptions	generate a STOP condition on I2C bus
Precondition	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

i2c_data_transmit

The description of i2c_data_transmit is shown as below:

Table 3-327. Function i2c_data_transmit

Function name	i2c_data_transmit
Function prototype	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
Function descriptions	I2C transmit data
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
data	transmit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x80);
```

i2c_data_receive

The description of i2c_data_receive is shown as below:

Table 3-328. Function i2c_data_receive

Function name	i2c_data_receive
---------------	------------------

Function prototype	uint32_t i2c_data_receive(uint32_t i2c_periph);
Function descriptions	I2C receive data
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	0x0000..0x00FF

Example:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

i2c_reload_enable

The description of i2c_reload_enable is shown as below:

Table 3-329. Function i2c_reload_enable

Function name	i2c_reload_enable
Function prototype	void i2c_reload_enable(uint32_t i2c_periph);
Function descriptions	enable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

i2c_reload_disable

The description of i2c_reload_disable is shown as below:

Table 3-330. Function i2c_reload_disable

Function name	i2c_reload_disable
Function prototype	void i2c_reload_disable(uint32_t i2c_periph);
Function descriptions	disable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C reload mode */
i2c_reload_disable(I2C0);
```

i2c_transfer_byte_number_config

The description of i2c_transfer_byte_number_config is shown as below:

Table 3-331. Function i2c_transfer_byte_number_config

Function name	i2c_transfer_byte_number_config
Function prototype	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint8_t byte_number);
Function descriptions	configure number of bytes to be transferred
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
byte_number	0x0-0xFF, number of bytes to be transferred
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

i2c_dma_enable

The description of i2c_dma_enable is shown as below:

Table 3-332. Function i2c_dma_enable

Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
Function descriptions	enable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
dma	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

i2c_dma_disable

The description of i2c_dma_disable is shown as below:

Table 3-333. Function i2c_dma_disable

Function name	i2c_dma_disable
Function prototype	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
Function descriptions	disable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
dma	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

i2c_pec_transfer

The description of i2c_pec_transfer is shown as below:

Table 3-334. Function i2c_pec_transfer

Function name	i2c_pec_transfer
Function prototype	void i2c_pec_transfer(uint32_t i2c_periph);
Function descriptions	I2C transfers PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

i2c_pec_enable

The description of i2c_pec_enable is shown as below:

Table 3-335. Function i2c_pec_enable

Function name	i2c_pec_enable
Function prototype	void i2c_pec_enable(uint32_t i2c_periph);
Function descriptions	enable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C PEC calculation */
```

```
i2c_pec_enable(I2C0);
```

i2c_pec_disable

The description of i2c_pec_disable is shown as below:

Table 3-336. Function i2c_pec_disable

Function name	i2c_pec_disable
Function prototype	void i2c_pec_disable(uint32_t i2c_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable(I2C0);
```

i2c_pec_value_get

The description of i2c_pec_value_get is shown as below:

Table 3-337. Function i2c_pec_value_get

Function name	i2c_pec_value_get
Function prototype	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
Function descriptions	get packet error checking value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	PEC value

Example:

```
/* I2C0 get packet error checking value */
```

```
uint32_t pec_value;
```

```
pec_value = i2c_pec_value_get(I2C0);
```

i2c_smbus_alert_enable

The description of i2c_smbus_alert_enable is shown as below:

Table 3-338. Function i2c_smbus_alert_enable

Function name	i2c_smbus_alert_enable
Function prototype	void i2c_smbus_alert_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Alert
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Alert */
```

```
i2c_smbus_alert_enable(I2C0);
```

i2c_smbus_alert_disable

The description of i2c_smbus_alert_disable is shown as below:

Table 3-339. Function i2c_smbus_alert_disable

Function name	i2c_smbus_alert_disable
Function prototype	void i2c_smbus_alert_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus Alert
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Alert */
```

```
i2c_smbus_alert_disable(I2C0);
```

i2c_smbus_default_addr_enable

The description of i2c_smbus_default_addr_enable is shown as below:

Table 3-340. Function i2c_smbus_default_addr_enable

Function name	i2c_smbus_default_addr_enable
Function prototype	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus device default address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C0);
```

i2c_smbus_default_addr_disable

The description of i2c_smbus_default_addr_disable is shown as below:

Table 3-341. Function i2c_smbus_default_addr_disable

Function name	i2c_smbus_default_addr_disable
Function prototype	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus device default address
Precondition	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

i2c_smbus_host_addr_enable

The description of i2c_smbus_host_addr_enable is shown as below:

Table 3-342. Function i2c_smbus_host_addr_enable

Function name	i2c_smbus_host_addr_enable
Function prototype	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Host address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

i2c_smbus_host_addr_disable

The description of i2c_smbus_host_addr_disable is shown as below:

Table 3-343. Function i2c_smbus_host_addr_disable

Function name	i2c_smbus_host_addr_disable
Function prototype	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus Host address

Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

i2c_extented_clock_timeout_enable

The description of i2c_extented_clock_timeout_enable is shown as below:

Table 3-344. Function i2c_extented_clock_timeout_enable

Function name	i2c_extented_clock_timeout_enable
Function prototype	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);
Function descriptions	enable extended clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

i2c_extented_clock_timeout_disable

The description of i2c_extented_clock_timeout_disable is shown as below:

Table 3-345. Function i2c_extented_clock_timeout_disable

Function name	i2c_extented_clock_timeout_disable
Function prototype	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);

Function descriptions	disable extended clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable(I2C0);
```

i2c_clock_timeout_enable

The description of i2c_clock_timeout_enable is shown as below:

Table 3-346. Function i2c_clock_timeout_enable

Function name	i2c_clock_timeout_enable
Function prototype	void i2c_clock_timeout_enable(uint32_t i2c_periph);
Function descriptions	enable clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

i2c_clock_timeout_disable

The description of i2c_clock_timeout_disable is shown as below:

Table 3-347. Function i2c_clock_timeout_disable

Function name	i2c_clock_timeout_disable
----------------------	---------------------------

Function prototype	void i2c_clock_timeout_disable(uint32_t i2c_periph);
Function descriptions	disable clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable(I2C0);
```

i2c_bus_timeout_b_config

The description of i2c_bus_timeout_b_config is shown as below:

Table 3-348. Function i2c_bus_timeout_b_config

Function name	i2c_bus_timeout_b_config
Function prototype	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure bus timeout B
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
timeout	0-0xffff, bus timeout B
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config(I2C0, 0xff);
```

i2c_bus_timeout_a_config

The description of i2c_bus_timeout_a_config is shown as below:

Table 3-349. Function i2c_bus_timeout_a_config

Function name	i2c_bus_timeout_a_config
Function prototype	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure bus timeout A
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
timeout	0-0xffff, bus timeout A
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config (I2C0, 0xff);
```

i2c_idle_clock_timeout_config

The description of i2c_idle_clock_timeout_config is shown as below:

Table 3-350. Function i2c_idle_clock_timeout_config

Function name	i2c_idle_clock_timeout_config
Function prototype	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure idle clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
timeout	bus timeout A
BUSTOA_DETECT_SCL_LOW	BUSTOA is used to detect SCL low timeout
BUSTOA_DETECT_IDLE	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config (I2C0, BUSTOA_DETECT_SCL_LOW);
```

i2c_flag_get

The description of i2c_flag_get is shown as below:

Table 3-351. Function i2c_flag_get

Function name	i2c_flag_get
Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
flag	I2C flags
<i>I2C_FLAG_TBE</i>	I2C_TDATA is empty during transmitting
<i>I2C_FLAG_TI</i>	transmit interrupt
<i>I2C_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_TC</i>	transfer complete in master mode
<i>I2C_FLAG_TCR</i>	transfer complete reload
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overflow/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TR</i>	whether the I2C is a transmitter or a receiver in slave mode
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_TBE);
```

i2c_flag_clear

The description of i2c_flag_clear is shown as below:

Table 3-352. Function i2c_flag_clear

Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
flag	I2C flags
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

i2c_interrupt_enable

The description of i2c_interrupt_enable is shown as below:

Table 3-353. Function i2c_interrupt_enable

Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);

Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
interrupt	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

i2c_interrupt_disable

The description of i2c_interrupt_disable is shown as below:

Table 3-354. Function i2c_interrupt_disable

Function name	i2c_interrupt_disable
Function prototype	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
interrupt	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt

<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

i2c_interrupt_flag_get

The description of i2c_interrupt_flag_get is shown as below:

Table 3-355. Function i2c_interrupt_flag_get

Function name	i2c_interrupt_flag_get
Function prototype	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
int_flag	I2C interrupt flags, refer to Table 3-294. i2c_interrupt_flag_enum .
<i>I2C_INT_FLAG_TI</i>	transmit interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_TC</i>	transfer complete in master mode interrupt flag
<i>I2C_INT_FLAG_TCR</i>	transfer complete reload interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE</i>	PEC error interrupt flag

<i>RR</i>	
<i>I2C_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>UT</i>	
<i>I2C_INT_FLAG_SMBAL</i>	SMBus Alert interrupt flag
<i>LT</i>	
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

i2c_interrupt_flag_clear

The description of i2c_interrupt_flag_clear is shown as below:

Table 3-356. Function i2c_interrupt_flag_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
int_flag	I2C interrupt flags, refer to Table 3-294. i2c_interrupt_flag_enum .
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD</i> <i>ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC</i> <i>RR</i>	PEC error interrupt flag

<i>I2C_INT_FLAG_TIMEO</i> <i>UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

3.15. PKCAU

The Public Key Cryptographic Acceleration Unit (PKCAU) can accelerate RSA (Rivest, Shamir and Adleman), Diffie-Hellmann (DH key exchange) and ECC (elliptic curve cryptography) in GF(p) (Galois domain). The PKCAU registers are listed in chapter [3.15.1](#), the PKCAU firmware functions are introduced in chapter [3.15.2](#).

3.15.1. Descriptions of Peripheral registers

PKCAU registers are listed in the table shown as below:

Table 3-357. PKCAU Registers

Registers	Descriptions
PKCAU_CTL	Control register
PKCAU_STAT	Status register
PKCAU_STATC	Status clear register

3.15.2. Descriptions of Peripheral functions

PKCAU firmware functions are listed in the table shown as below:

Table 3-358. PKCAU firmware function

Function name	Function description
pkcau_deinit	reset PKCAU
pkcau_mont_struct_para_init	initialize montgomery parameter structure with a default value
pkcau_mod_struct_para_init	initialize modular parameter structure with a default value
pkcau_mod_exp_struct_para_init	initialize modular exponentation parameter structure with a default value
pkcau_arithmetic_struct_para_init	initialize arithmetic parameter structure with a default value
pkcau_crt_struct_para_init	initialize CRT parameter structure with a default value

Function name	Function description
pkcau_ec_group_struct_para_init	initialize ECC curve parameter structure with a default value
pkcau_point_struct_para_init	initialize point parameter structure with a default value
pkcau_signature_struct_para_init	initialize signature parameter structure with a default value
pkcau_hash_struct_para_init	initialize hash parameter structure with a default value
pkcau_ecc_out_struct_para_init	initialize ecc output parameter structure with a default value
pkcau_enable	enable PKCAU
pkcau_disable	disable PKCAU
pkcau_start	start operation
pkcau_mode_set	configure the PKCAU operation mode
pkcau_mont_param_operation	execute montgomery parameter operation
pkcau_mod_operation	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
pkcau_mod_exp_operation	execute modular exponentiation operation
pkcau_mod_inver_operation	execute modular inversion operation
pkcau_mod_reduc_operation	execute modular reduction operation
pkcau_arithmetic_operation	execute arithmetic addition operation
pkcau_crt_exp_operation	execute RSA CRT exponentiation operation
pkcau_point_check_operation	execute point check operation
pkcau_point_mul_operation	execute point multiplication operation
pkcau_ecdsa_sign_operation	execute ECDSA sign operation
pkcau_ecdsa_verification_operation	execute ECDSA verify operation
pkcau_flag_get	get PKCAU flag status
pkcau_flag_clear	clear PKCAU flag status
pkcau_interrupt_enable	enable PKCAU interrupt
pkcau_interrupt_disable	disable PKCAU interrupt
pkcau_interrupt_flag_get	get PKCAU interrupt flag status
pkcau_interrupt_flag_clear	clear PKCAU interrupt flag status

Structure pkcau_mont_parameter_struct

Table 3-359. Structure pkcau_mont_parameter_struct

Member name	Function description
modulus_n	modulus value n
modulus_n_len	modulus length in byte

Structure pkcau_mod_parameter_struct

Table 3-360. Structure pkcau_mod_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
opr_d_b	operand B

Member name	Function description
opr_d_b_len	operand b length in byte
modulus_n	modulus value n
modulus_n_len	modulus length in byte

Structure pkcau_mod_exp_parameter_struct

Table 3-361. Structure pkcau_mod_exp_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
exp_e	exponent e
e_len	exponent length in byte
modulus_n	modulus n
modulus_n_len	modulus length in byte
mont_para	montgomery parameter R2 mod n
mont_para_len	montgomery parameter length in byte

Structure pkcau_arithmetic_parameter_struct

Table 3-362. Structure pkcau_arithmetic_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	length of operand A in byte
opr_d_b	operand B
opr_d_b_len	length of operand B in byte

Structure pkcau_crt_parameter_struct

Table 3-363. Structure pkcau_crt_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	length of operand A in byte
opr_d_dp	operand dp
opr_d_dp_len	length of operand dp in byte
opr_d_dq	operand dq
opr_d_dq_len	length of operand dq in byte
opr_d_qinv	operand qinv
opr_d_qinv_len	length of operand qinv in byte
opr_d_p	prime operand p
opr_d_p_len	length of operand p in byte
opr_d_q	prime operand q
opr_d_q_len	length of operand q in byte

Structure pkcau_ec_group_parameter_struct

Table 3-364. Structure pkcau_ec_group_parameter_struct

Member name	Function description
modulus_p	curve modulus p
modulus_p_len	curve modulus p length in byte
coff_a	curve coefficient a
coff_a_len	curve coefficient a length in byte
coff_b	curve coefficient b
coff_b_len	curve coefficient b length in byte
base_point_x	curve base point coordinate x
base_point_x_len	curve base point coordinate x length in byte
base_point_y	curve base point coordinate y
base_point_y_len	curve base point coordinate y length in byte
order_n	curve prime order n
order_n_len	curve prime order n length in byte
a_sign	curve coefficient a sign
multi_k	scalar multiplier k
multi_k_len	length of scalar multiplier k
integer_k	integer k
integer_k_len	integer k length in byte
private_key_d	private key d
private_key_d_len	private key d length in byte
mont_para	montgomery parameter R2 mod n
mont_para_len	montgomery parameter R2 mod n length in byte

Structure pkcau_point_parameter_struct

Table 3-365. Structure pkcau_point_parameter_struct

Member name	Function description
point_x	point coordinate x
point_x_len	point coordinate x length in byte
point_y	point coordinate y
point_y_len	point coordinate y length in byte

Structure pkcau_signature_parameter_struct

Table 3-366. Structure pkcau_signature_parameter_struct

Member name	Function description
sign_r	signature part r
sign_r_len	signature part r length in byte
sign_s	signature part s
sign_s_len	signature part s length in byte

Structure pkcau_hash_parameter_struct

Table 3-367. Structure pkcau_hash_parameter_struct

Member name	Function description
hash_z	hash value z
hash_z_len	hash value z length in byte

Structure pkcau_ecc_out_struct

Table 3-368. Structure pkcau_ecc_out_struct

Member name	Function description
sign_extra	flag of extended ECDSA sign (extra outputs)
sign_r	signature part r
sign_s	signature part s
point_x	point coordinate x
point_y	point coordinate y

pkcau_deinit

The description of pkcau_deinit is shown as below:

Table 3-369. Function pkcau_deinit

Function name	pkcau_deinit
Function prototype	void pkcau_deinit(void);
Function descriptions	reset PKCAU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PKCAU */
pkcau_deinit();
```

pkcau_mont_struct_para_init

The description of pkcau_mont_struct_para_init is shown as below:

Table 3-370. Function pkcau_mont_struct_para_init

Function name	pkcau_mont_struct_para_init
Function prototype	void pkcau_mont_struct_para_init(pkcau_mont_parameter_struct*)

	init_para);
Function descriptions	initialize montgomery parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	montgomery parameter struct, the structure members can refer to Table 3-359. Structure pkcau_mont_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU montgomery parameter struct with a default value */
```

```
pkcau_mont_parameter_struct pkcau_mont_parameter;
```

```
pkcau_mont_struct_para_init(&pkcau_mont_parameter);
```

pkcau_mod_struct_para_init

The description of pkcau_mod_struct_para_init is shown as below:

Table 3-371. Function pkcau_mod_struct_para_init

Function name	pkcau_mod_struct_para_init
Function prototype	void pkcau_mod_struct_para_init(pkcau_mod_parameter_struct* init_para);
Function descriptions	initialize modular parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular parameter struct, the structure members can refer to Table 3-360. Structure pkcau_mod_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU modular parameter struct with a default value */
```

```
pkcau_mod_parameter_struct pkcau_mod_parameter;
```

```
pkcau_mod_struct_para_init(&pkcau_mod_parameter);
```

pkcau_mod_exp_struct_para_init

The description of pkcau_mod_exp_struct_para_init is shown as below:

Table 3-372. Function pkcau_mod_exp_struct_para_init

Function name	pkcau_mod_exp_struct_para_init
Function prototype	void pkcau_mod_exp_struct_para_init(pkcau_mod_exp_parameter_struct* init_para);
Function descriptions	initialize modular exponentiation parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular exponentiation parameter struct, the structure members can refer to Table 3-361. Structure pkcau_mod_exp_parameter_struct .
Return value	
-	-

Example:

```
/* initialize PKCAU modular exponentiation parameter struct with a default value */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);
```

pkcau_arithmetic_struct_para_init

The description of pkcau_arithmetic_struct_para_init is shown as below:

Table 3-373. Function pkcau_arithmetic_struct_para_init

Function name	pkcau_arithmetic_struct_para_init
Function prototype	void pkcau_arithmetic_struct_para_init(pkcau_arithmetic_parameter_struct* init_para);
Function descriptions	initialize arithmetic parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	arithmetic parameter struct, the structure members can refer to Table 3-362. Structure pkcau_arithmetic_parameter_struct .
Return value	
-	-

Example:

```
/* initialize PKCAU arithmetic parameter struct with a default value */
```

```
pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;
```

```
pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);
```

pkcau_crt_struct_para_init

The description of pkcau_crt_struct_para_init is shown as below:

Table 3-374. Function pkcau_crt_struct_para_init

Function name	pkcau_crt_struct_para_init
Function prototype	void pkcau_crt_struct_para_init(pkcau_crt_parameter_struct* init_para);
Function descriptions	initialize CRT parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	CRT parameter struct, the structure members can refer to Table 3-363. Structure pkcau_crt_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU CRT parameter struct with a default value */
```

```
pkcau_crt_parameter_struct pkcau_crt_parameter;
```

```
pkcau_crt_struct_para_init(&pkcau_crt_parameter);
```

pkcau_ec_group_struct_para_init

The description of pkcau_ec_group_struct_para_init is shown as below:

Table 3-375. Function pkcau_ec_group_struct_para_init

Function name	pkcau_ec_group_struct_para_init
Function prototype	void pkcau_ec_group_struct_para_init(pkcau_ec_group_parameter_struct* init_para);
Function descriptions	initialize ECC curve parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	ECC curve parameter struct, the structure members can refer to Table 3-364. Structure pkcau_ec_group_parameter_struct.

Return value	
-	-

Example:

```
/* initialize PKCAU ECC curve parameter struct with a default value */
```

```
pkcau_ec_group_parameter_struct pkcau_ec_group_parameter;
```

```
pkcau_ec_group_struct_para_init(&pkcau_ec_group_parameter);
```

pkcau_point_struct_para_init

The description of pkcau_point_struct_para_init is shown as below:

Table 3-376. Function pkcau_point_struct_para_init

Function name	pkcau_point_struct_para_init
Function prototype	void pkcau_point_struct_para_init(pkcau_point_parameter_struct* init_para);
Function descriptions	initialize point parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	point parameter struct, the structure members can refer to Table 3-365. Structure pkcau_point_parameter_struct .
Return value	
-	-

Example:

```
/* initialize PKCAU point parameter struct with a default value */
```

```
pkcau_point_parameter_struct pkcau_point_parameter;
```

```
pkcau_point_struct_para_init(&pkcau_point_parameter);
```

pkcau_signature_struct_para_init

The description of pkcau_signature_struct_para_init is shown as below:

Table 3-377. Function pkcau_signature_struct_para_init

Function name	pkcau_signature_struct_para_init
Function prototype	void pkcau_signature_struct_para_init(pkcau_signature_parameter_struct* init_para);
Function descriptions	initialize signature parameter structure with a default value
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
init_para	signature parameter struct, the structure members can refer to Table 3-366. Structure pkcau_signature_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU signature parameter struct with a default value */
pkcau_signature_parameter_struct pkcau_signature_parameter;
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
```

pkcau_hash_struct_para_init

The description of pkcau_hash_struct_para_init is shown as below:

Table 3-378. Function pkcau_hash_struct_para_init

Function name	pkcau_hash_struct_para_init
Function prototype	void pkcau_hash_struct_para_init(pkcau_hash_parameter_struct* init_para);
Function descriptions	initialize hash parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	hash parameter struct, the structure members can refer to Table 3-367. Structure pkcau_hash_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU hash parameter struct with a default value */
pkcau_hash_parameter_struct pkcau_hash_parameter;
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
```

pkcau_ecc_out_struct_para_init

The description of pkcau_ecc_out_struct_para_init is shown as below:

Table 3-379. Function pkcau_ecc_out_struct_para_init

Function name	pkcau_ecc_out_struct_para_init
---------------	--------------------------------

Function prototype	void pkcau_ecc_out_struct_para_init(pkcau_ecc_out_struct* init_para)
Function descriptions	initialize ECC out parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-368. Structure pkcau ecc out struct.
Return value	
-	-

Example:

```
/* initialize PKCAU modular reduction parameter struct with a default value */
pkcau_ecc_out_struct pkcau_ecc_out_parameter;
pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_parameter);
```

pkcau_enable

The description of pkcau_enable is shown as below:

Table 3-380. Function pkcau_enable

Function name	pkcau_enable
Function prototype	void pkcau_enable(void);
Function descriptions	enable PKCAU
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PKCAU */
pkcau_enable();
```

pkcau_disable

The description of pkcau_disable is shown as below:

Table 3-381. Function pkcau_disable

Function name	pkcau_disable
Function prototype	void pkcau_disable(void);
Function descriptions	disable PKCAU
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PKCAU */
pkcau_disable();
```

pkcau_start

The description of pkcau_start is shown as below:

Table 3-382. Function pkcau_start

Function name	pkcau_start
Function prototype	void pkcau_start(void);
Function descriptions	start operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start operation */
pkcau_start();
```

pkcau_mode_set

The description of pkcau_mode_set is shown as below:

Table 3-383. Function pkcau_mode_set

Function name	pkcau_mode_set
Function prototype	void pkcau_mode_set(uint32_t mode);
Function descriptions	configure the PKCAU operation mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	PKCAU operation mode
<i>PKCAU_MODE_MOD_EXP</i>	Montgomery parameter computation then modular exponentiation
<i>PKCAU_MODE_MONT_PARAM</i>	Montgomery parameter computation only
<i>PKCAU_MODE_MOD_EXP_FAST</i>	modular exponentiation only
<i>PKCAU_MODE_CRT_EXP</i>	RSA CRT exponentiation
<i>PKCAU_MODE_MOD_INVERSION</i>	modular inversion
<i>PKCAU_MODE_ARITHMETIC_ADD</i>	arithmetic addition
<i>PKCAU_MODE_ARITHMETIC_SUB</i>	arithmetic subtraction
<i>PKCAU_MODE_ARITHMETIC_MUL</i>	arithmetic multiplication
<i>PKCAU_MODE_ARITHMETIC_COMP</i>	arithmetic comparison
<i>PKCAU_MODE_MOD_REDUCTION</i>	modular reduction
<i>PKCAU_MODE_MOD_ADD</i>	modular addition
<i>PKCAU_MODE_MOD_SUB</i>	modular subtraction
<i>PKCAU_MODE_MONT_MUL</i>	Montgomery multiplication
<i>PKCAU_MODE_ECC_MUL</i>	Montgomery parameter computation then ECC scalar multiplication
<i>PKCAU_MODE_ECC_MUL_FAST</i>	ECC scalar multiplication only
<i>PKCAU_MODE_ECDSA_A_SIGN</i>	ECDSA sign
<i>PKCAU_MODE_ECDSA_A_VERIFICATION</i>	ECDSA verification
<i>PKCAU_MODE_POINT</i>	point on elliptic curve Fp check

<code>_CHECK</code>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PKCAU operation mode as PKCAU_MODE_ARITHMETIC_ADD */
pkcau_mode_set(PKCAU_MODE_ARITHMETIC_ADD);
```

pkcau_mont_param_operation

The description of pkcau_mont_param_operation is shown as below:

Table 3-384. Function pkcau_mont_param_operation

Function name	pkcau_mont_param_operation
Function prototype	void pkcau_mont_param_operation(pkcau_mont_parameter_struct* mont_para, uint8_t* results)
Function descriptions	execute montgomery parameter operation
Precondition	-
The called functions	-
Input parameter{in}	
mont_para	montgomery parameter struct, the structure members can refer to Table 3-359. Structure pkcau_mont_parameter_struct .
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t results[ME_MOD_SIZE];

/* initialize the montgomery parameter structure */
pkcau_mont_parameter_struct pkcau_mont_parameter;
pkcau_mont_struct_para_init(&pkcau_mont_parameter);

/* initialize the montgomery parameters */
pkcau_mont_parameter.modulus_n_len = ME_MOD_SIZE;
pkcau_mont_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */
pkcau_mont_param_operation(&pkcau_mont_parameter, results);
```

pkcau_mod_operation

The description of pkcau_mod_operation is shown as below:

Table 3-385. Function pkcau_mod_operation

Function name	pkcau_mod_operation
Function prototype	pkcau_mod_operation(pkcau_mod_parameter_struct* mod_para, uint32_t mode, uint8_t* results)
Function descriptions	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
Precondition	-
The called functions	-
Input parameter{in}	
mod_para	modular parameter struct, the structure members can refer to Table 3-360. Structure pkcau_mod_parameter_struct.
Input parameter{in}	
mode	modular operation mode
<i>PKCAU_MODE_MOD_ADD</i>	modular addition
<i>PKCAU_MODE_MOD_SUB</i>	modular subtraction
<i>PKCAU_MODE_MONT_MUL</i>	Montgomery multiplication
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t results[MA_MOD_SIZE];

/* initialize the modular parameter structure */

pkcau_mod_parameter_struct pkcau_mod_parameter;
pkcau_mod_struct_para_init(&pkcau_mod_parameter);

/* initialize the modular parameters */

pkcau_mod_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_parameter.oprd_b = (uint8_t *)oprd_b;
pkcau_mod_parameter.modulus_n_len = MA_MOD_SIZE;
pkcau_mod_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular addition operation */
```

```
pkcau_mod_operation(&pkcau_mod_parameter, PKCAU_MODE_MOD_ADD, results);
```

pkcau_mod_exp_operation

The description of pkcau_mod_exp_operation is shown as below:

Table 3-386. Function pkcau_mod_exp_operation

Function name	pkcau_mod_exp_operation
Function prototype	pkcau_mod_exp_operation(pkcau_mod_exp_parameter_struct* mod_exp_para, uint32_t mode, uint8_t* results)
Function descriptions	execute modular exponentiation operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_exp_para	modular exponentiation parameter struct, the structure members can refer to Table 3-361. Structure pkcau_mod_exp_parameter_struct .
Input parameter{in}	
mode	modular exponentiation operation mode
<i>PKCAU_MODE_MOD_EXP</i>	montgomery parameter computation then modular exponentiation
<i>PKCAU_MODE_MOD_EXP_FAST</i>	modular exponentiation only
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t mod_exp_res[ME_MOD_SIZE];

/* initialize the modular exponentiation parameter structure */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);

/* initialize the modular exponentiation parameters */
pkcau_mod_exp_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_exp_parameter.oprd_a_len = sizeof(oprd_a);
pkcau_mod_exp_parameter.exp_e = (uint8_t *)exp_e;
pkcau_mod_exp_parameter.e_len = ME_E_SIZE;
pkcau_mod_exp_parameter.modulus_n_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
```

```
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;

pkcau_mod_exp_parameter.mont_para_len = sizeof(mont_para);

/* execute modular exponentation fast operation */

pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,
PKCAU_MODE_MOD_EXP_FAST, mod_exp_res);
```

pkcau_mod_inver_operation

The description of pkcau_mod_inver_operation is shown as below:

Table 3-387. Function pkcau_mod_inver_operation

Function name	pkcau_mod_inver_operation
Function prototype	pkcau_mod_inver_operation(pkcau_mod_parameter_struct* mod_inver_para, uint8_t* results)
Function descriptions	execute modular inversion operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_inver_para	modular inversion parameter struct, the structure members can refer to Table 3-360. Structure pkcau_mod_parameter_struct .
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t mod_inver_res[ME_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);

/* initialize the modular parameters */

pkcau_mod_inver_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_inver_parameter.oprd_a = sizeof(oprd_a);

pkcau_mod_inver_parameter.modulus_n_len = ME_MOD_SIZE;

pkcau_mod_inver_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular inversion operation */

pkcau_mod_inver_operation(&pkcau_mod_inver_parameter, mod_inver_res);
```

pkcau_mod_reduc_operation

The description of pkcau_mod_reduc_operation is shown as below:

Table 3-388. Function pkcau_mod_reduc_operation

Function name	pkcau_mod_reduc_operation
Function prototype	void pkcau_mod_reduc_operation(pkcau_mod_parameter_struct* mod_reduc_para, uint8_t* results)
Function descriptions	execute modular reduction operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_reduc_para	modular reduction parameter struct, the structure members can refer to Table 3-360. Structure pkcau_mod_parameter_struct .
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t mod_reduc_res[MA_MOD_SIZE];

/* initialize the modular inversion parameter structure */
pkcau_mod_parameter_struct pkcau_mod_reduc_parameter;
pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);

/* initialize the modular parameters */
pkcau_mod_reduc_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_reduc_parameter.oprd_a_len = MA_A_SIZE;
pkcau_mod_reduc_parameter.modulus_n_len = MA_MOD_SIZE;
pkcau_mod_reduc_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular reduction operation */
pkcau_mod_reduc_operation(&pkcau_mod_reduc_parameter, mod_reduc_res);
```

pkcau_arithmetic_operation

The description of pkcau_arithmetic_operation is shown as below:

Table 3-389. Function pkcau_arithmetic_operation

Function name	pkcau_arithmetic_operation
Function prototype	void pkcau_arithmetic_operation(pkcau_arithmetic_parameter_struct* arithmetic_para, uint32_t mode, uint8_t* results)

Function descriptions	execute arithmetic operation
Precondition	-
The called functions	-
Input parameter{in}	
arithmetic_para	arithmetic parameter struct, the structure members can refer to Table 3-362. Structure pkcau_arithmetic_parameter_struct.
Input parameter{in}	
mode	arithmetic operation mode
<i>PKCAU_MODE_ARITHMETIC_ADD</i>	arithmetic addition
<i>PKCAU_MODE_ARITHMETIC_SUB</i>	arithmetic subtraction
<i>PKCAU_MODE_ARITHMETIC_MUL</i>	arithmetic multiplication
<i>PKCAU_MODE_ARITHMETIC_COMP</i>	arithmetic comparison
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```

uint8_t ari_multi_res[AM_SIZE];

/* initialize the arithmetic parameter structure */
pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;
pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);

/* initialize the arithmetic addition parameters */
pkcau_ari_multi_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_ari_multi_parameter.oprd_a_len = AM_A_SIZE;
pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;
pkcau_ari_multi_parameter.oprd_b_len = AM_B_SIZE;

/* execute arithmetic addition operation */
pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD, ari_multi_res);

```

pkcau_crt_exp_operation

The description of pkcau_crt_exp_operation is shown as below:

Table 3-390. Function pkcau_crt_exp_operation

Function name	pkcau_crt_exp_operation
Function prototype	void pkcau_crt_exp_operation(pkcau_crt_parameter_struct* crt_para, uint8_t* results);
Function descriptions	execute RSA CRT exponentiation operation
Precondition	-
The called functions	-
Input parameter{in}	
crt_para	CRT parameter struct, the structure members can refer to Table 3-363. Structure pkcau_crt_parameter_struct.
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t crt_res[sizeof(rsa_crt_a)];

/* initialize the arithmetic parameter structure */
pkcau_crt_parameter_struct pkcau_crt_parameter;
pkcau_crt_exp_operation(&pkcau_crt_parameter);

/* initialize the input ECC curve parameters */
pkcau_crt_parameter.oprd_a = (uint8_t *)rsa_crt_a;
pkcau_crt_parameter.oprd_a_len = sizeof(rsa_crt_a);
pkcau_crt_parameter.oprd_dp = (uint8_t *)rsa_crt_dp;
pkcau_crt_parameter.oprd_dp_len = sizeof(rsa_crt_dp);
pkcau_crt_parameter.oprd_dq = (uint8_t *)rsa_crt_dq;
pkcau_crt_parameter.oprd_dq_len = sizeof(rsa_crt_dq);
pkcau_crt_parameter.oprd_qinv = (uint8_t *)rsa_crt_qinv;
pkcau_crt_parameter.oprd_qinv_len = sizeof(rsa_crt_qinv);
pkcau_crt_parameter.oprd_p = (uint8_t *)rsa_crt_p;
pkcau_crt_parameter.oprd_p_len = sizeof(rsa_crt_p);
pkcau_crt_parameter.oprd_q = (uint8_t *)rsa_crt_q;
pkcau_crt_parameter.oprd_q_len = sizeof(rsa_crt_q);

/* execute RSA CRT exponentiation operation */
```

```
pkcau crt_exp_operation(&pkcau crt_parameter, crt_res);
```

pkcau_point_check_operation

The description of pkcau_point_check_operation is shown as below:

Table 3-391. Function pkcau_point_check_operation

Function name	pkcau_point_check_operation
Function prototype	uint8_t pkcau_point_check_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para);
Function descriptions	execute point check operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-365. Structure pkcau_point_parameter_struct .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-364. Structure pkcau_ec_group_parameter_struct .
Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the point is on an elliptic curve or not

Example:

```
uint8_t res = 1;

/* initialize point parameter and ECC curve parameter structure */
pkcau_point_parameter_struct pkcau_point_parameter;
pkcau_ec_group_parameter_struct pkcau_curve_group;
pkcau_point_struct_para_init(&pkcau_point_parameter);
pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */
pkcau_point_check_parameter.point_x = pcheck_x;
pkcau_point_check_parameter.point_x_len = sizeof(pcheck_x);
pkcau_point_check_parameter.point_y = pcheck_y;
pkcau_point_check_parameter.point_y_len = sizeof(pcheck_y);

/* initialize the input ECC curve parameter */
pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;
```

```
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);

pkcau_curve_group.coff_b = (uint8_t *)brainpoolp256r1_b;

pkcau_curve_group.coff_b_len = sizeof(brainpoolp256r1_b);

pkcau_curve_group.a_sign = 0;

/* execute point check operation */

res = pkcau_point_check_operation(&pkcau_point_parameter, &pkcau_curve_group);
```

pkcau_point_mul_operation

The description of pkcau_point_mul_operation is shown as below:

Table 3-392. Function pkcau_point_mul_operation

Function name	pkcau_point_mul_operation
Function prototype	void pkcau_point_mul_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para, \n uint32_t mode, pkcau_ecc_out_struct* result);
Function descriptions	execute point multiplication operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-365. Structure pkcau_point_parameter_struct.
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-364. Structure pkcau_ec_group_parameter_struct.
Input parameter{in}	
mode	point multiplication operation mode
PKCAU_MODE_ECC_MUL	montgomery parameter computation then ECC scalar multiplication
PKCAU_MODE_ECC_MUL_FAST	ECC scalar multiplication only
Output parameter{out}	
result	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-368. Structure pkcau_ecc_out_struct.
Return value	
-	-

Example:

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.point_x = res_x;

pkcau_ecc_out_result.point_y = res_y;

/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_parameter.point_x = ec_pmul_x;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

pkcau_point_parameter.point_y = ec_pmul_y;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_a_len    = sizeof(brainpoolp256r1_a);

pkcau_curve_group.a_sign        = 0;

pkcau_curve_group.multi_k       = ec_pmul_k;

pkcau_curve_group.multi_k_len   = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL, pkcau_ecc_out_result);
```

pkcau_ecdsa_sign_operation

The description of pkcau_ecdsa_sign_operation is shown as below:

Table 3-393. Function pkcau_ecdsa_sign_operation

Function name	pkcau_ecdsa_sign_operation
Function prototype	uint8_t pkcau_ecdsa_sign_operation(pkcau_hash_parameter_struct* hash_para, \ const pkcau_ec_group_parameter_struct* curve_group_para, \ pkcau_ecc_out_struct* result);
Function descriptions	execute ECDSA sign operation
Precondition	-
The called functions	-
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to Table 3-367. Structure pkcau_hash_parameter_struct.
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-364. Structure pkcau_ec_group_parameter_struct.
Output parameter{out}	
result	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-368. Structure pkcau_ecc_out_struct.
Return value	
uint8_t	flag indicating whether the signature operation was successful

Example:

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.sign_r = r;

pkcau_ecc_out_result.sign_s = s;

/* initialize hash parameter and ECC curve parameter structure */

pkcau_hash_parameter_struct pkcau_hash_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input hash parameter */

pkcau_hash_parameter.hash_z      = hash;

pkcau_hash_parameter.hash_z_len = DATA_SIZE;

/* initialize the input ECC curve parameter */
```

```

pkcau_curve_group.modulus_p      = secp112r2_p;

pkcau_curve_group.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.coff_a         = secp112r2_a;

pkcau_curve_group.coff_a_len     = sizeof(secp112r2_a);

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.base_point_x   = secp112r2_gx;

pkcau_curve_group.base_point_x_len = sizeof(secp112r2_gx);

pkcau_curve_group.base_point_y   = secp112r2_gy;

pkcau_curve_group.base_point_y_len = sizeof(secp112r2_gy);

pkcau_curve_group.order_n        = secp112r2_n,

pkcau_curve_group.order_n_len    = sizeof(secp112r2_n);

pkcau_curve_group.integer_k      = k;

pkcau_curve_group.integer_k_len  = DATA_SIZE;

pkcau_curve_group.private_key_d  = d;

pkcau_curve_group.private_key_d_len = DATA_SIZE;

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(&pkcau_hash_parameter, &pkcau_curve_group, results);

```

pkcau_ecdsa_verification_operation

The description of pkcau_ecdsa_verification_operation is shown as below:

Table 3-394. Function pkcau_ecdsa_verification_operation

Function name	pkcau_ecdsa_verification_operation
Function prototype	uint8_t pkcau_ecdsa_verification_operation(pkcau_point_parameter_struct* point_para, pkcau_hash_parameter_struct* hash_para, pkcau_signature_parameter_struct* signature_para, const pkcau_ec_group_parameter_struct* curve_group_para)
Function descriptions	execute ECDSA verification operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-365. Structure pkcau_point_parameter_struct .
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to Table 3-367.

	<u>Structure pkcau_hash_parameter_struct.</u>
Input parameter{in}	
signature_para	signature parameter struct, the structure members can refer to <u>Table 3-366.</u> <u>Structure pkcau_signature_parameter_struct.</u>
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to <u>Table 3-364.</u> <u>Structure pkcau_ec_group_parameter_struct.</u>
Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the signature verification operation was successful

Example:

```
uint8_t verify_res = 1;

/* ECC curve parameter structure */
pkcau_ec_group_parameter_struct pkcau_curve_group;

/* hash parameter structure */
pkcau_hash_parameter_struct pkcau_hash_parameter;

/* signature parameter structure */
pkcau_signature_parameter_struct pkcau_signature_parameter;

/* point parameter structure */
pkcau_point_parameter_struct pkcau_point_parameter;

/* initialize the ECC curve parameter, hash parameter, point parameter and signature
parameter structure */
pkcau_ec_group_struct_para_init(&pkcau_curve_group);
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
pkcau_point_struct_para_init(&pkcau_point_parameter);
pkcau_signature_struct_para_init(&pkcau_signature_parameter);

/* initialize the input ECC signature parameters */
pkcau_signature_parameter.sign_r      = (uint8_t *)ecc_verify_r;
pkcau_signature_parameter.sign_r_len = sizeof(ecc_verify_r);
pkcau_signature_parameter.sign_s      = (uint8_t *)ecc_verify_s;
pkcau_signature_parameter.sign_s_len = sizeof(ecc_verify_s);

/* initialize the input point parameters */
```



```

pkcau_point_parameter.point_x      = ecc_verify_x;

pkcau_point_parameter.point_x_len = sizeof(ecc_verify_x);

pkcau_point_parameter.point_y      = ecc_verify_y;

pkcau_point_parameter.point_y_len = sizeof(ecc_verify_y);

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p        = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len    = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a           = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len       = sizeof(brainpoolp256r1_a);
pkcau_curve_group.a_sign           = 0;
pkcau_curve_group.base_point_x     = (uint8_t *)brainpoolp256r1_gx;
pkcau_curve_group.base_point_x_len = sizeof(brainpoolp256r1_gx);
pkcau_curve_group.base_point_y     = (uint8_t *)brainpoolp256r1_gy;
pkcau_curve_group.base_point_y_len = sizeof(brainpoolp256r1_gy);
pkcau_curve_group.order_n          = (uint8_t *)brainpoolp256r1_n;
pkcau_curve_group.order_n_len      = sizeof(brainpoolp256r1_n);

/* initialize the input hash parameters */

pkcau_hash_parameter.hash_z        = (uint8_t *)ecc_verify_hash;
pkcau_hash_parameter.hash_z_len    = sizeof(ecc_verify_hash);

/* execute ECDSA verification operation */

verify_res = pkcau_ecdsa_verification_operation(&pkcau_point_parameter,
&pkcau_hash_parameter, &pkcau_signature_parameter, &pkcau_curve_group);

```

pkcau_flag_get

The description of pkcau_flag_get is shown as below:

Table 3-395. Function pkcau_flag_get

Function name	pkcau_flag_get
Function prototype	FlagStatus pkcau_flag_get(uint32_t flag);
Function descriptions	get PKCAU flag status
Precondition	-
The called functions	-
Input parameter{in}	

flag	PKCAU flags
<i>PKCAU_FLAG_ADDRE</i> <i>RR</i>	address error flag
<i>PKCAU_FLAG_RAME</i> <i>RR</i>	PKCAU RAM error flag
<i>PKCAU_FLAG_END</i>	end of PKCAU operation flag
<i>PKCAU_FLAG_BUSY</i>	busy flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get PKCAU flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = pkcau_flag_get(PKCAU_FLAG_ADDRERR);
```

pkcau_flag_clear

The description of pkcau_flag_clear is shown as below:

Table 3-396. Function pkcau_flag_clear

Function name	pkcau_flag_clear
Function prototype	void pkcau_flag_clear(uint32_t flag);
Function descriptions	clear PKCAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	PKCAU flags
<i>PKCAU_FLAG_ADDRE</i> <i>RR</i>	address error flag
<i>PKCAU_FLAG_RAME</i> <i>RR</i>	PKCAU RAM error flag
<i>PKCAU_FLAG_END</i>	end of PKCAU operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear address error flag*/
```

```
pkcau_flag_clear(PKCAU_FLAG_ADDRERR);
```

pkcau_interrupt_enable

The description of pkcau_interrupt_enable is shown as below:

Table 3-397. Function pkcau_interrupt_enable

Function name	pkcau_interrupt_enable
Function prototype	void pkcau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable PKCAU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type
<i>PKCAU_INT_ADDRER R</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PKCAU address error interrupt */
pkcau_interrupt_enable(PKCAU_INT_ADDRERR);
```

pkcau_interrupt_disable

The description of pkcau_interrupt_disable is shown as below:

Table 3-398. Function pkcau_interrupt_disable

Function name	pkcau_interrupt_disable
Function prototype	void pkcau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable PKCAU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type
<i>PKCAU_INT_ADDRER R</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable PKCAU address error interrupt */
pkcau_interrupt_disable(PKCAU_INT_ADDRERR);
```

pkcau_interrupt_flag_get

The description of pkcau_interrupt_flag_get is shown as below:

Table 3-399. Function pkcau_interrupt_flag_get

Function name	pkcau_interrupt_flag_get
Function prototype	FlagStatus pkcau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get PKCAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_ADDRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_END</i>	end of PKCAU operation interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get PKCAU interrupt flag status */
FlagStatus flag_state = RESET;
flag_state = pkcau_interrupt_flag_get(PKCAU_INT_FLAG_ADDRERR);
```

pkcau_interrupt_flag_clear

The description of pkcau_interrupt_flag_clear is shown as below:

Table 3-400. Function pkcau_interrupt_flag_clear

Function name	pkcau_interrupt_flag_clear
Function prototype	void pkcau_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear PKCAU flag interrupt status
Precondition	-

The called functions	-
Input parameter{in}	
int_flag	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_ADDRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_END</i>	end of PKCAU operation interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear address error interrupt flag*/
```

```
pkcau_interrupt_flag_clear(PKCAU_INT_FLAG_ADDRERR);
```

3.16. PMU

According to the Power management unit (PMU), provides six types of power saving modes, including Sleep, Deep-sleep, Standby mode, SRAM_sleep, Wi-Fi_sleep and BLE_sleep mode. The PMU registers are listed in chapter [3.16.1](#), the PMU firmware functions are introduced in chapter [3.16.2](#).

3.16.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-401. PMU Registers

Registers	Descriptions
PMU_CTL0	PMU control register 0
PMU_CS0	PMU control and status register 0
PMU_CTL1	PMU control register 1
PMU_CS1	PMU control and status register 1
PMU_PAR0	PMU parameter register 0
PMU_PAR1	PMU parameter register 1
PMU_PAR2	PMU parameter register 2
PMU_RFCTL	PMU RF control register
PMU_RFPAR	PMU RF timer parameter register
PMU_INTF	PMU BLE interrupt flag register
PMU_INTEN	PMU BLE interrupt enable register

Registers	Descriptions
PMU_INTC	PMU BLE interrupt clear register

3.16.2. Descriptions of Peripheral functions

PMU firmware functions are listed in the table shown as below:

Table 3-402. PMU firmware function

Function name	Function description
pmu_deinit	reset PMU register
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU LVD
pmu_backup_write_enable	enable write access to the registers in backup domain
pmu_backup_write_disable	disable write access to the registers in backup domain
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work in deep-sleep mode
pmu_to_standbymode	PMU work in standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_wifi_power_enable	enable WIFI power
pmu_wifi_power_disable	disable WIFI power
pmu_wifi_sram_control	WIFI & SRAM low power control
pmu_ble_control	BLE low power control
pmu_ble_wakeup_request_enable	enable BLE wakeup request
pmu_ble_wakeup_request_disable	disable BLE wakeup request
pmu_pll_force_enable	MCU PLL power force open/close
pmu_pll_force_disable	disable MCU PLL power open/close force
pmu_ble_rf_config	configure BLE RF sequence
pmu_rf_force_enable	enable RF sequence force open/close
pmu_rf_force_disable	disable RF sequence open/close force
pmu_rf_sequence_config	configure RF sequence
pmu_flag_get	get flag status
pmu_flag_clear	clear flag bit
pmu_interrupt_enable	enable PMU interrupt
pmu_interrupt_disable	disable PMU interrupt
pmu_interrupt_flag_get	get PMU interrupt flag
pmu_interrupt_flag_clear	clear PMU interrupt flag

pmu_deinit

The description of pmu_deinit is shown as below:

Table 3-403. Function pmu_deinit

Function name	pmu_deinit
---------------	------------

Function prototype	void pmu_deinit(void);
Function descriptions	reset PMU register
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
```

```
pmu_deinit();
```

pmu_lvd_select

The description of pmu_lvd_select is shown as below:

Table 3-404. Function pmu_lvd_select

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_t_n	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.1V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	voltage threshold is 3.1V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select(PMU_LVDT_5);
```

pmu_lvd_disable

The description of pmu_lvd_disable is shown as below:

Table 3-405. Function pmu_lvd_disable

Function name	pmu_lvd_disable
Function prototype	void pmu_lvd_disable (void);
Function descriptions	disable PMU lvd
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU lvd */
pmu_lvd_disable ();
```

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-406. Function pmu_backup_write_enable

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable write access to the registers in backup domain
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to the registers in backup domain */
pmu_backup_write_enable();
```


pmu_backup_write_disable

The description of pmu_backup_write_disable is shown as below:

Table 3-407. Function pmu_backup_write_disable

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable(void);
Function descriptions	disable write access to the registers in backup domain
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write access to the registers in backup domain */
pmu_backup_write_disable();
```

pmu_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

Table 3-408. Function pmu_to_sleepmode

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
sleepmodecmd	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at sleep mode */
pmu_to_sleepmode(WFI_CMD);
```

pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-409. Function pmu_to_deepsleepmode

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
Function descriptions	PMU work in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO operates normally when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
lowdrive	Low-driver mode
<i>PMU_LOWDRIVER_DISABLE</i>	Low-driver mode disable in deep-sleep mode
<i>PMU_LOWDRIVER_ENABLE</i>	Low-driver mode enable in deep-sleep mode
Input parameter{in}	
deepsleepmodecmd	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work in deep-sleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDO_NORMAL, PMU_LOWDRIVER_DISABLE, WFI_CMD);
```

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-410. Function pmu_to_standbymode

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(uint8_t standbymodecmd);
Function descriptions	PMU work in standby mode
Precondition	-

The called functions	-
Input parameter{in}	
standbymodecmd	command to enter standby mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at standby mode */
pmu_to_standbymode(WFI_CMD);
```

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

Table 3-411. Function pmu_wakeup_pin_enable

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(void);
Function descriptions	enable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_pin	WKUP pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PA15)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PA7)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup pin 0 */
pmu_wakeup_pin_enable(PMU_WAKEUP_PIN0);
```

pmu_wakeup_pin_disable

The description of pmu_wakeup_pin_disable is shown as below:

Table 3-412. Function pmu_wakeup_pin_disable

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable(void);
Function descriptions	disable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_pin	WKUP pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PA15)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PA7)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup pin 0 */
pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);
```

pmu_wifi_power_enable

The description of pmu_wifi_power_enable is shown as below:

Table 3-413. Function pmu_wifi_power_enable

Function name	pmu_wifi_power_enable
Function prototype	void pmu_wifi_power_enable(void);
Function descriptions	enable WIFI power
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable WIFI power */
pmu_wifi_power_enable();
```

pmu_wifi_power_disable

The description of pmu_wifi_power_disable is shown as below:

Table 3-414. Function pmu_wifi_power_disable

Function name	pmu_wifi_power_disable
Function prototype	void pmu_wifi_power_disable(void);
Function descriptions	disable WIFI power
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable WIFI power */
pmu_wifi_power_disable();
```

pmu_wifi_sram_control

The description of pmu_wifi_sram_control is shown as below:

Table 3-415. Function pmu_wifi_sram_control

Function name	pmu_wifi_sram_control
Function prototype	void pmu_wifi_sram_control(uint32_t wifi_sram);
Function descriptions	WIFI & SRAM low power control
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_sram	low power control
<i>PMU_WIFI_SLEEP</i>	WIFI go to sleep
<i>PMU_WIFI_WAKE</i>	WIFI wakeup
<i>PMU_SRAM1_SLEEP</i>	SRAM1 go to sleep
<i>PMU_SRAM1_WAKE</i>	SRAM1 wakeup
<i>PMU_SRAM2_SLEEP</i>	SRAM2 go to sleep
<i>PMU_SRAM2_WAKE</i>	SRAM2 wakeup
<i>PMU_SRAM3_SLEEP</i>	SRAM3 go to sleep
<i>PMU_SRAM3_WAKE</i>	SRAM3 wakeup
<i>PMU_SRAM0_SLEEP</i>	SRAM0 go to sleep
<i>PMU_SRAM0_WAKE</i>	SRAM0 wakeup
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* WIFI & SRAM low power control */
pmu_wifi_sram_control(PMU_WIFI_SLEEP);
```

pmu_ble_control

The description of pmu_ble_control is shown as below:

Table 3-416. Function pmu_ble_control

Function name	pmu_ble_control
Function prototype	void pmu_ble_control(uint32_t wifi_sram);
Function descriptions	BLE low power control
Precondition	-
The called functions	-
Input parameter{in}	
ble	BLE low power control
<i>PMU_BLE_SLEEP</i>	BLE go to sleep
<i>PMU_BLE_WAKE</i>	BLE wakeup
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* BLE low power control */
pmu_ble_control(PMU_BLE_SLEEP);
```

pmu_ble_wakeup_request_enable

The description of pmu_ble_wakeup_request_enable is shown as below:

Table 3-417. Function pmu_ble_wakeup_request_enable

Function name	pmu_ble_wakeup_request_enable
Function prototype	void pmu_ble_wakeup_request_enable(void);
Function descriptions	enable BLE wakeup request
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable BLE wakeup request */
pmu_ble_wakeup_request_enable();
```

pmu_ble_wakeup_request_disable

The description of pmu_ble_wakeup_request_disable is shown as below:

Table 3-418. Function pmu_ble_wakeup_request_disable

Function name	pmu_ble_wakeup_request_disable
Function prototype	void pmu_ble_wakeup_request_disable(void);
Function descriptions	disable BLE wakeup request
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable BLE wakeup request */
pmu_ble_wakeup_request_disable();
```

pmu_pll_force_enable

The description of pmu_pll_force_enable is shown as below:

Table 3-419. Function pmu_pll_force_enable

Function name	pmu_pll_force_enable
Function prototype	void pmu_pll_force_enable(uint32_t force);
Function descriptions	enable MCU PLL power force open/close
Precondition	-
The called functions	-
Input parameter{in}	
force	PLL power force open/close
<i>PMU_PLL_FORCE_OP</i>	Software force open, open MCU PLL power

<i>EN</i>	
<i>PMU_PLL_FORCE_CL</i> <i>OSE</i>	Software force close, close MCU PLL power
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable MCU PLL power force open/close */
pmu_pll_force_enable(PMU_PLL_FORCE_OPEN);
```

pmu_pll_force_disable

The description of pmu_pll_force_disable is shown as below:

Table 3-420. Function pmu_pll_force_disable

Function name	pmu_pll_force_disable
Function prototype	void pmu_pll_force_disable(uint32_t force);
Function descriptions	disable MCU PLL power open/close force
Precondition	-
The called functions	-
Input parameter{in}	
force	PLL power force open/close
<i>PMU_PLL_FORCE_OP</i> <i>EN</i>	Software force open, open MCU PLL power
<i>PMU_PLL_FORCE_CL</i> <i>OSE</i>	Software force close, close MCU PLL power
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable MCU PLL power force open/close */
pmu_pll_force_disable(PMU_PLL_FORCE_CLOSE);
```

pmu_ble_rf_config

The description of pmu_ble_rf_config is shown as below:

Table 3-421. Function pmu_ble_rf_config

Function name	pmu_ble_rf_config
Function prototype	void pmu_ble_rf_config(uint32_t mode);

Function descriptions	configure BLE RF sequence
Precondition	-
The called functions	-
Input parameter{in}	
mode	enable RF mode
<i>PMU_BLE_RF_SOFTWARE</i>	enable RF by software
<i>PMU_BLE_RF_HARDWARE</i>	enable RF BLE hardware
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RF by software */
pmu_ble_rf_config(PMU_BLE_RF_SOFTWARE);
```

pmu_rf_force_enable

The description of pmu_rf_force_enable is shown as below:

Table 3-422. Function pmu_rf_force_enable

Function name	pmu_rf_force_enable
Function prototype	void pmu_rf_force_enable(uint32_t force);
Function descriptions	enable RF sequence force open/close
Precondition	-
The called functions	-
Input parameter{in}	
force	RF sequence force open/close
<i>PMU_RF_FORCE_OPEN</i>	Software force start, open RF power
<i>PMU_RF_FORCE_CLOSE</i>	Software force close, close RF power
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RF sequence force open/close */
pmu_rf_force_enable(PMU_RF_FORCE_OPEN);
```

pmu_rf_force_disable

The description of pmu_rf_force_disable is shown as below:

Table 3-423. Function pmu_rf_force_disable

Function name	pmu_rf_force_disable
Function prototype	void pmu_rf_force_disable(uint32_t force);
Function descriptions	disable RF sequence open/close force
Precondition	-
The called functions	-
Input parameter{in}	
force	RF sequence force open/close
<i>PMU_RF_FORCE_OPEN</i>	Software force start, open RF power
<i>PMU_RF_FORCE_CLOSE</i>	Software force close, close RF power
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RF sequence open/close force */
pmu_rf_force_disable(PMU_RF_FORCE_OPEN);
```

pmu_rf_sequence_config

The description of pmu_rf_sequence_config is shown as below:

Table 3-424. Function pmu_rf_sequence_config

Function name	pmu_rf_sequence_config
Function prototype	void pmu_rf_sequence_config(uint32_t mode);
Function descriptions	configure RF sequence
Precondition	-
The called functions	-
Input parameter{in}	
mode	RF sequence mode
<i>PMU_RF_SOFTWARE</i>	RF software sequence enable
<i>PMU_RF_HARDWARE</i>	RF hardware sequence enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure RF sequence */
```

```
pmu_rf_sequence_config(PMU_RF_SOFTWARE);
```

pmu_flag_get

The description of pmu_flag_get is shown as below:

Table 3-425. Function pmu_flag_get

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t flag);
Function descriptions	get flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	lvd flag
PMU_FLAG_LDOVSRF	LDO voltage select ready flag
PMU_FLAG_LDRF	low-driver mode ready flag
PMU_FLAG_WIFI_SLEEP	WIFI is in sleep state
PMU_FLAG_WIFI_ACTIVE	WIFI is in active state
PMU_FLAG_BLE_POWER	BLE power status
PMU_FLAG_SRAM1_SLEEP	SRAM1 is in sleep state
PMU_FLAG_SRAM1_ACTIVE	SRAM1 is in active state
PMU_FLAG_SRAM0_SLEEP	SRAM0 is in sleep state
PMU_FLAG_SRAM0_ACTIVE	SRAM0 is in active state
PMU_FLAG_SRAM2_SLEEP	SRAM2 is in sleep state
PMU_FLAG_SRAM2_ACTIVE	SRAM2 is in active state
PMU_FLAG_SRAM3_SLEEP	SRAM3 is in sleep state
PMU_FLAG_SRAM3_ACTIVE	SRAM3 is in active state

<i>PMU_FLAG_BLE_SLEEP</i>	BLE is in sleep state
<i>PMU_FLAG_BLE_ACTIVE</i>	BLE is in active state
<i>PMU_FLAG_BLE_POWER_FALL</i>	BLE power status falling edge flag
<i>PMU_FLAG_BLE_POWER_RISE</i>	BLE power status rising edge flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag state */
```

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

pmu_flag_clear

The description of pmu_flag_clear is shown as below:

Table 3-426. Function pmu_flag_clear

Function name	pmu_flag_clear
Function prototype	void pmu_flag_clear(uint32_t flag);
Function descriptions	clear flag bit
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<i>PMU_FLAG_RESET_LDRF</i>	reset Low-driver mode ready flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear(PMU_FLAG_RESET_WAKEUP);
```

pmu_interrupt_enable

The description of pmu_interrupt_enable is shown as below:

Table 3-427. Function pmu_interrupt_enable

Function name	pmu_interrupt_enable
Function prototype	void pmu_interrupt_enable(uint32_t interrupt);
Function descriptions	enable PMU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	PMU interrupt
<i>PMU_INT_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PMU interrupt */
pmu_interrupt_enable(PMU_INT_BLE_POWER_FALL);
```

pmu_interrupt_disable

The description of pmu_interrupt_disable is shown as below:

Table 3-428. Function pmu_interrupt_disable

Function name	pmu_interrupt_disable
Function prototype	void pmu_interrupt_disable(uint32_t interrupt);
Function descriptions	disable PMU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	PMU interrupt
<i>PMU_INT_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable PMU interrupt */

pmu_interrupt_disable(PMU_INT_BLE_POWER_FALL);
```

pmu_interrupt_flag_get

The description of pmu_interrupt_flag_get is shown as below:

Table 3-429. Function pmu_interrupt_flag_get

Function name	pmu_interrupt_flag_get
Function prototype	FlagStatus pmu_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get PMU interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	PMU interrupt flag
<i>PMU_INT_FLAG_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_FLAG_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* get PMU interrupt flag */

pmu_interrupt_flag_get(PMU_INT_FLAG_BLE_POWER_FALL);
```

pmu_interrupt_flag_clear

The description of pmu_interrupt_flag_clear is shown as below:

Table 3-430. Function pmu_interrupt_flag_clear

Function name	pmu_interrupt_flag_clear
Function prototype	FlagStatus pmu_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear PMU interrupt flag
Precondition	-
The called functions	-

Input parameter{in}	
int_flag	PMU interrupt flag
<i>PMU_INT_FLAG_RESET_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_FLAG_RESET_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear PMU interrupt flag */
```

```
pmu_interrupt_flag_get(PMU_INT_FLAG_RESET_BLE_POWER_FALL);
```

3.17. QSPI

The QSPI is a specialized interface that communicate with Flash memories. This interface support single, dual or quad SPI FLASH. The QSPI registers are listed in chapter [3.17.1](#), the QSPI firmware functions are introduced in chapter [3.17.2](#).

3.17.1. Descriptions of Peripheral registers

QSPI registers are listed in the table shown as below:

Table 3-431. QSPI registers

Registers	Descriptions
QSPI_CTL	QSPI control register
QSPI_DCFG	QSPI device configuration register
QSPI_STAT	QSPI status register
QSPI_STATC	QSPI status clear register
QSPI_DTLEN	QSPI data length register
QSPI_TCFG	QSPI transfer configuration register
QSPI_ADDR	QSPI address register
QSPI_ALTE	QSPI alternate bytes register
QSPI_DATA	QSPI data register
QSPI_STATMK	QSPI status mask register
QSPI_STATMATCH	QSPI status match register
QSPI_INTERVAL	QSPI interval register
QSPI_TMOUT	QSPI timeout register

Registers	Descriptions
QSPI_FLUSH	QSPI fifo flush register

3.17.2. Descriptions of Peripheral functions

QSPI firmware functions are listed in the table shown as below:

Table 3-432. QSPI firmware function

Function name	Function description
qspi_deinit	reset QSPI peripheral
qspi_struct_para_init	initialize the parameters of QSPI init structure with the default values
qspi_cmd_struct_para_init	initialize the parameters of QSPI command structure with the default values
qspi_polling_struct_para_init	initialize the parameters of QSPI read polling structure with the default values
qspi_init	initialize QSPI
qspi_enable	enable QSPI
qspi_disable	disable QSPI
qspi_dma_enable	enable QSPI DMA
qspi_dma_disable	disable QSPI DMA
qspi_command_config	configure QSPI command parameters
qspi_polling_config	configure QSPI read polling mode
qspi_memorymapped_config	configure QSPI memory mapped mode
qspi_data_transmit	QSPI transmit data function
qspi_data_receive	QSPI receive data function
qspi_transmission_abort	abort the current transmission
qspi_flag_get	get QSPI flag status
qspi_flag_clear	clear QSPI flag status
qspi_interrupt_enable	enable QSPI interrupt
qspi_interrupt_disable	disable QSPI interrupt
qspi_interrupt_flag_get	get QSPI interrupt flag status
qspi_interrupt_flag_clear	clear QSPI interrupt flag status

Structure qspi_init_struct

Table 3-433. Structure qspi_init_struct

Member name	Function description
prescaler	QSPI prescaler
fifo_threshold	QSPI FIFO threshold
sample_shift	QSPI sample shift
flash_size	external flash size
cs_high_time	CLK cycles which the chip select (nCS) must stay high between two command

Member name	Function description
	sequences
clock_mode	QSPI clock mode

Structure qspi_command_struct

Table 3-434. Structure qspi_command_struct

Member name	Function description
instruction_mode	instruction mode
instruction	8 bits instruction
addr_mode	address mode
addr_size	address size
addr	address to be send to the external flash memory
altebytes_mode	alternate bytes mode
altebytes_size	alternate bytes size
altebytes	alternate bytes information
dummycycles	dummy cycles
data_mode	data mode
data_length	32 bits data length
sioo_mode	send instruction only once mode

Structure qspi_polling_struct

Table 3-435. Structure qspi_polling_struct

Member name	Function description
match	match value
mask	mask value
interval	number of clock cycles between two read during automatic polling phases
statusbytes_size	the size of the status bytes received
match_mode	method used for determining a match
polling_stop	if read polling is stopped after a match

qspi_deinit

The description of qspi_deinit is shown as below:

Table 3-436. Function qspi_deinit

Function name	qspi_deinit
Function prototype	void qspi_deinit(void);
Function descriptions	reset QSPI peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset QSPI */
qspi_deinit();
```

qspi_struct_para_init

The description of qspi_struct_para_init is shown as below:

Table 3-437. Function qspi_struct_para_init

Function name	qspi_struct_para_init
Function prototype	void qspi_struct_para_init(qspi_init_struct *init_para);
Function descriptions	initialize the parameters of QSPI structure with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	QSPI parameter structure, the structure members can refer to Table 3-433. Structure qspi_init_struct.
Return value	
-	-

Example:

```
/* initialize the parameters of QSPI */
qspi_init_struct qspi_init_para;
qspi_struct_para_init(&qspi_init_para);
```

qspi_cmd_struct_para_init

The description of qspi_cmd_struct_para_init is shown as below:

Table 3-438. Function qspi_cmd_struct_para_init

Function name	qspi_cmd_struct_para_init
Function prototype	void qspi_cmd_struct_para_init(qspi_command_struct *init_para);
Function descriptions	initialize the parameters of QSPI command structure with the default values
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
init_para	QSPI command parameter structure, the structure members can refer to Table 3-434. Structure qspi_command_struct.
Return value	
-	-

Example:

```
/* initialize the parameters of QSPI command structrue */
```

```
qspi_command_struct qspi_cmd_para;
```

```
qspi_cmd_struct_para_init(&qspi_cmd_para);
```

qspi_polling_struct_para_init

The description of qspi_polling_struct_para_init is shown as below:

Table 3-439. Function qspi_polling_struct_para_init

Function name	qspi_polling_struct_para_init
Function prototype	void qspi_polling_struct_para_init(qspi_polling_struct *init_para);
Function descriptions	initialize the parameters of QSPI read polling structure with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	QSPI read polling parameter structure, the structure members can refer to Table 3-435. Structure qspi_polling_struct.
Return value	
-	-

Example:

```
/* initialize the parameters of QSPI read polling structrue */
```

```
qspi_polling_struct qspi_polling_para;
```

```
qspi_polling_struct_para_init(&qspi_polling_para);
```

qspi_init

The description of qspi_init is shown as below:

Table 3-440. Function qspi_init

Function name	qspi_init
Function prototype	void qspi_init(qspi_init_struct* qspi_struct);

Function descriptions	Initialize QSPI
Precondition	-
The called functions	-
Input parameter{in}	
qspi_struct	QSPI parameter initialization structure, the structure members can refer to members of the structure Table 3-433. Structure qspi_init_struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the init parameter structure */

qspi_struct_para_init(&qspi_init_para);

qspi_init_para.clock_mode = QSPI_CLOCK_MODE_0;

qspi_init_para.fifo_threshold = 4;

qspi_init_para.sample_shift = QSPI_SAMPLE_SHIFTING_HALFCYCLE;

qspi_init_para.cs_high_time = QSPI_CS_HIGH_TIME_2_CYCLE;

qspi_init_para.flash_size = 27;

qspi_init_para.prescaler = 4;

qspi_init(&qspi_init_para);

```

qspi_enable

The description of qspi_enable is shown as below:

Table 3-441. Function qspi_enable

Function name	qspi_enable
Function prototype	void qspi_enable(void);
Function descriptions	enable QSPI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable QSPI */
```

```
qspi_enable();
```

qspi_disable

The description of qspi_disable is shown as below:

Table 3-442. Function qspi_disable

Function name	qspi_disable
Function prototype	void qspi_disable(void);
Function descriptions	disable QSPI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable QSPI */
```

```
qspi_disable();
```

qspi_dma_enable

The description of qspi_dma_enable is shown as below:

Table 3-443. Function qspi_dma_enable

Function name	qspi_dma_enable
Function prototype	void qspi_dma_enable(void);
Function descriptions	enable QSPI DMA
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable QSPI DMA */
```

```
qspi_dma_enable();
```

qspi_dma_disable

The description of qspi_dma_disable is shown as below:

Table 3-444. Function qspi_dma_disable

Function name	qspi_dma_disable
Function prototype	void qspi_dma_disable(void);
Function descriptions	disable QSPI DMA
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable QSPI DMA */
```

```
qspi_dma_disable();
```

qspi_data_length_config

The description of qspi_data_length_config is shown as below:

Table 3-445. Function qspi_data_length_config

Function name	qspi_data_length_config
Function prototype	void qspi_data_length_config(uint32_t dtlen);
Function descriptions	configure QSPI data length
Precondition	-
The called functions	-
Input parameter{in}	
dtlen	QSPI data length (1 ~ 4294967295 (4G-1))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure QSPI data length */
```

```
qspi_data_length_config(8);
```

qspi_command_config

The description of qspi_command_config is shown as below:

Table 3-446. Function qspi_command_config

Function name	qspi_command_config
Function prototype	void qspi_command_config(qspi_command_struct *cmd);
Function descriptions	configure QSPI command parameters
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command parameter structure, the structure members can refer to Table 3-434. Structure qspi_command_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* write enable */
qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;
qspi_cmd.instruction = WRITE_ENABLE_CMD;
qspi_cmd.addr_mode = QSPI_ADDR_NONE;
qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_NONE;
qspi_cmd.data_mode = QSPI_DATA_NONE;
qspi_cmd.dummycycles = 0;
qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;
qspi_command_config(&qspi_cmd);

```

qspi_polling_config

The description of qspi_polling_config is shown as below:

Table 3-447. Function qspi_polling_config

Function name	qspi_polling_config
Function prototype	void qspi_polling_config(qspi_command_struct *cmd, qspi_polling_struct

	*cfg);
Function descriptions	configure QSPI read polling mode
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command parameter structure, the structure members can refer to Table 3-434. Structure qspi_command_struct.
Input parameter{in}	
cfg	QSPI read polling parameter structure, the structure members can refer to Table 3-435. Structure qspi_polling_struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* initialize the QSPI read polling parameter structure */
qspi_polling_struct_para_init(&polling_cmd);

/* read enable */
polling_cmd.match = 0x02;
polling_cmd.mask = 0x02;
polling_cmd.match_mode = QSPI_MATCH_MODE_AND;
polling_cmd.statusbytes_size = 1;
polling_cmd.interval = 0x10;
polling_cmd.polling_stop = QSPI_POLLING_STOP_ENABLE;
qspi_cmd.instruction = READ_STATUS_REG1_CMD;
qspi_cmd.data_mode = QSPI_DATA_1_LINE;
qspi_polling_config(&qspi_cmd, &polling_cmd);

```

qspi_memorymapped_config

The description of qspi_memorymapped_config is shown as below:

Table 3-448. Function qspi_memorymapped_config

Function name	qspi_memorymapped_config
----------------------	--------------------------

Function prototype	void qspi_memorymapped_config(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
Function descriptions	configure QSPI memory mapped mode
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command parameter structure, the structure members can refer to Table 3-434. Structure qspi_command_struct .
Input parameter{in}	
timeout	0-0xFFFF
Input parameter{in}	
toen	timeout counter
QSPI_TMOUT_DISABLE	disable timeout counter
QSPI_TMOUT_ENABLE	enable timeout counter
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the QSPI command parameter structure */

qspi_cmd_struct_para_init(&qspi_cmd);

qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;

qspi_cmd.instruction = RDID;

qspi_cmd.addr_mode = QSPI_ADDR_NONE;

qspi_cmd.addr_size = QSPI_ADDR_8_BITS;

qspi_cmd.addr = 0;

qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_1_LINE;

qspi_cmd.altebytes_size = QSPI_ALTE_BYTES_24_BITS;

qspi_cmd.altebytes = 0;

qspi_cmd.dummycycles = 0;

qspi_cmd.data_mode = QSPI_DATA_1_LINE;

qspi_cmd.data_length = 3;

qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;

```

```
qspi_memorymapped_config(&qspi_cmd, 0xf, QSPI_TMOU_ENABLE);
```

qspi_data_transmit

The description of qspi_data_transmit is shown as below:

Table 3-449. Function qspi_data_transmit

Function name	qspi_data_transmit
Function prototype	void qspi_data_transmit(uint8_t *tdata);
Function descriptions	QSPI transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
tdata	pointer to the data to be transmitted
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* QSPI transmit data */
uint8_t data[32];
qspi_data_transmit(&data);
```

qspi_data_receive

The description of qspi_data_receive is shown as below:

Table 3-450. Function qspi_data_receive

Function name	qspi_data_receive
Function prototype	void qspi_data_receive(uint8_t *rdata);
Function descriptions	QSPI receive data function
Precondition	-
The called functions	-
Input parameter{in}	
rdata	pointer to the data to be received
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* QSPI receive data */
```

```
uint8_t data[32];

qspi_data_receive(&data);
```

qspi_transmission_abort

The description of qspi_transmission_abort is shown as below:

Table 3-451. Function qspi_transmission_abort

Function name	qspi_transmission_abort
Function prototype	void qspi_transmission_abort(void);
Function descriptions	abort the current transmission
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* abort QSPI transmission */

qspi_transmission_abort();
```

qspi_flag_get

The description of qspi_flag_get is shown as below:

Table 3-452. Function qspi_flag_get

Function name	qspi_flag_get
Function prototype	FlagStatus qspi_flag_get(uint32_t flag);
Function descriptions	get QSPI flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	QSPI flag
QSPI_FLAG_BUSY	busy flag
QSPI_FLAG_TERR	transfer error flag
QSPI_FLAG_TC	transfer complete flag
QSPI_FLAG_FT	FIFO threshold flag
QSPI_FLAG_RPMF	read polling match flag
QSPI_FLAG_TMOUT	timeout flag
Output parameter{out}	

-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

qspi_flag_clear

The description of qspi_flag_clear is shown as below:

Table 3-453. Function qspi_flag_clear

Function name	qspi_flag_clear
Function prototype	void qspi_flag_clear(uint32_t flag);
Function descriptions	clear QSPI flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	QSPI flag
QSPI_FLAG_TERR	transfer error flag
QSPI_FLAG_TC	transfer complete flag
QSPI_FLAG_RPMF	read polling match flag
QSPI_FLAG_TMOU	timeout flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

qspi_interrupt_enable

The description of qspi_interrupt_enable is shown as below:

Table 3-454. Function qspi_interrupt_enable

Function name	qspi_interrupt_enable
Function prototype	void qspi_interrupt_enable(uint32_t interrupt);
Function descriptions	enable QSPI interrupt
Precondition	-

The called functions	-
Input parameter{in}	
interrupt	QSPI interrupt
<i>QSPI_INT_TC</i>	transfer complete interrupt
<i>QSPI_INT_FT</i>	FIFO threshold interrupt
<i>QSPI_INT_TERR</i>	transfer error interrupt
<i>QSPI_INT_RPMF</i>	read polling match interrupt
<i>QSPI_INT_TMOUT</i>	timeout interrupt
Output parameter{out}	
-	-
Return value	

Example:

```
/* enable QSPI transfer complete interrupt */
qspi_interrupt_enable(QSPI_INT_TC);
```

qspi_interrupt_disable

The description of qspi_interrupt_disable is shown as below:

Table 3-455. Function qspi_interrupt_disable

Function name	qspi_interrupt_disable
Function prototype	void qspi_interrupt_disable(uint8_t interrupt);
Function descriptions	disable QSPI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	QSPI interrupt
<i>QSPI_INT_TC</i>	transfer complete interrupt
<i>QSPI_INT_FT</i>	FIFO threshold interrupt
<i>QSPI_INT_TERR</i>	transfer error interrupt
<i>QSPI_INT_RPMF</i>	read polling match interrupt
<i>QSPI_INT_TMOUT</i>	timeout interrupt
Output parameter{out}	
-	-
Return value	

Example:

```
/* disable QSPI transfer complete interrupt */
qspi_interrupt_disable(QSPI_INT_TC);
```

qspi_interrupt_flag_get

The description of qspi_interrupt_flag_get is shown as below:

Table 3-456. Function qspi_interrupt_flag_get

Function name	qspi_interrupt_flag_get
Function prototype	FlagStatus qspi_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get QSPI interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	QSPI interrupt flag
QSPI_INT_FLAG_TRANSFER	transfer error interrupt flag
QSPI_INT_FLAG_TC	transfer complete interrupt flag
QSPI_INT_FLAG_FT	FIFO threshold interrupt flag
QSPI_INT_FLAG_RPMF	read polling match interrupt flag
QSPI_INT_FLAG_TMOU	timeout interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

qspi_interrupt_flag_clear

The description of qspi_interrupt_flag_clear is shown as below:

Table 3-457. Function qspi_interrupt_flag_clear

Function name	qspi_interrupt_flag_clear
Function prototype	void qspi_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear QSPI interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	QSPI interrupt flag
QSPI_INT_FLAG_TRANSFER	transfer error interrupt flag

<i>R</i>	
QSPI_INT_FLAG_TC	transfer complete interrupt flag
QSPI_INT_FLAG_RPM <i>F</i>	read polling match interrupt flag
QSPI_INT_FLAG_TMO <i>UT</i>	timeout interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

3.18. RCU

RCU is the reset and clock unit. Reset Control includes the control of three kinds of reset: power reset, system reset and backup domain reset. The Clock Control unit provides a range of frequencies and clock functions. The RCU registers are listed in chapter [3.18.1](#), the RCU firmware functions are introduced in chapter [3.18.2](#).

3.18.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

Table 3-458. RCU Registers

Registers	Descriptions
RCU_CTL	Control register
RCU_PLL	PLL register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB2 reset register
RCU_AHB3RST	AHB3 reset register
RCU_APB1RST	APB1 reset register
RCU_APB2RST	APB2 reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_AHB3EN	AHB3 enable register
RCU_APB1EN	APB1 enable register
RCU_APB2EN	APB2 enable register

Registers	Descriptions
RCU_AHB1SPEN	AHB1 sleep mode enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_PLLDIGCFG0	PLLDIG clock configuration register 0
RCU_CFG1	Clock configuration register 1
RCU_ADDCTL	Additional clock control register
RCU_PLLDIGCFG1	PLLDIG clock configuration register 1
RCU_VKEY	Voltage key register
RCU_DSV	Deep-sleep mode voltage register

3.18.2. Descriptions of Peripheral functions

Table 3-459. RCU firmware function

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_irc16m_dfs_to_rf_enable	enable differential signal of IRC16M to RF module
rcu_irc16m_dfs_to_rf_disable	disable differential signal of IRC16M to RF module
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_fmc_clock_sleep_enable	enabled FMC clock when sleep mode
rcu_fmc_clock_sleep_disable	disabled FMC clock when sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_rfpll_cal_enable	enable the RF PLL calculation
rcu_rfpll_cal_disable	disable the RF PLL calculation
rcu_control_unit_powerup	power up rcu control unit
rcu_control_unit_powerdown	power down rcu control unit
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source and divider
rcu_ckout1_config	configure the CK_OUT1 clock source and divider
rcu_plldig_config	configure the PLLDIG output clock frequency
rcu_plldigdiv_sys_config	configure PLLDIG clock divider factor for system clock
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_rtc_div_config	configure the frequency division of RTC clock when HXTAL was selected as its clock source

Function name	Function description
rcu_trng_div_config	configure the frequency division of the TRNG source clock
rcu_i2c0_clock_config	configure the I2C0 clock source selection
rcu_usart0_clock_config	configure the USART0 clock source selection
rcu_irc16m_div_config	configure IRC16M clock divider factor for system clock
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode
rcu_rf_hxtal_clock_monitor_enable	enable the RF HXTAL clock monitor
rcu_rf_hxtal_clock_monitor_disable	disable the RF HXTAL clock monitor
rcu_irc16m_adjust_value_set	set the IRC16M adjust value
rcu_voltage_key_unlock	unlock the voltage key
rcu_deepsleep_voltage_set	select deep-sleep mode voltage
rcu_clock_freq_get	get the system clock, bus clock frequency

Enum rcu_periph_enum

Table 3-460. Enum rcu_periph_enum

enum name	Function description
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_CRC	CRC clock
RCU_WIFI	WIFI clock
RCU_WIFIRUN	WIFIRUN clock
RCU_SRAM0	SRAM0 clock
RCU_SRAM1	SRAM1 clock
RCU_SRAM2	SRAM2 clock
RCU_SRAM3	SRAM3 clock
RCU_DMA	DMA clock
RCU_BLE	BLE clock
RCU_PKCAU	PKCAU clock

enum name	Function description
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock
RCU_QSPI	QSPI clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER5	TIMER5 clock
RCU_WWDGT	WWDGT clock
RCU_RFI	RFI clock
RCU_UART1	UART1 clock
RCU_USART0	USART0 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_PMU	PMU clock
RCU_RTC	RTC clock
RCU_TIMER0	TIMER0 clock
RCU_UART2	UART0 clock
RCU_ADC	ADC clock
RCU_SPI	SPI clock
RCU_SYSCFG	SYSCFG clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_RF	RF clock

Enum rcu_periph_reset_enum

Table 3-461. Enum rcu_periph_reset_enum

enum name	Function description
RCU_GPIOARST	GPIOA clock reset
RCU_GPIOBRST	GPIOB clock reset
RCU_GPIOCRST	GPIOC clock reset
RCU_CRCRST	CRC clock reset
RCU_WIFIRST	WIFI clock reset
RCU_DMARST	DMA clock reset
RCU_BLERST	BLE clock reset
RCU_PKCAURST	PKCAU clock reset
RCU_CAURST	CAU clock reset
RCU_HAURST	HAU clock reset
RCU_TRNGRST	TRNG clock reset
RCU_QSPIRST	QSPI clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset

enum name	Function description
RCU_TIMER5RST	TIMER5 clock reset
RCU_WWDGTRST	WWDGT clock reset
RCU_RFIRST	RFI clock reset
RCU_UART1RST	UART1 clock reset
RCU_USART0RST	USART0 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_PMURST	PMU clock reset
RCU_TIMER0RST	TIMER0 clock reset
RCU_UART2RST	UART2 clock reset
RCU_ADCRST	ADC clock reset
RCU_SPIRST	SPI clock reset
RCU_SYSCFGRST	SYSCFG clock reset
RCU_TIMER15RST	TIMER15 clock reset
RCU_TIMER16RST	TIMER16 clock reset
RCU_RFRST	RF clock reset

Enum rcu_unit_enum

Table 3-462. Enum rcu_unit_enum

enum name	Function description
RCU_UNIT_HXTAL	HXTAL
RCU_UNIT_PLLDIG	PLLDIG
RCU_UNIT_RFPLL	RFPLL
RCU_UNIT_LDOANA	LDOANA
RCU_UNIT_LDOCLK	LDOCLK
RCU_UNIT_BANDGAP	BANDGAP

Enum rcu_flag_enum

Table 3-463. Enum rcu_flag_enum

enum name	Function description
RCU_FLAG_IRC16MS TB	IRC16M stabilization flags
RCU_FLAG_HXTALST B	HXTAL stabilization flags
RCU_FLAG_PLLDIGST B	PLLDIG stabilization flags
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC32KST B	IRC32K stabilization flags

enum name	Function description
RCU_FLAG_EPRST	external PIN reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	software reset flags
RCU_FLAG_FWDGTR ST	FWDGT reset flags
RCU_FLAG_WWDGTR ST	WWDGT reset flags
RCU_FLAG_LPRST	low-power reset flags

Enum rcu_int_flag_enum

Table 3-464. Enum rcu_int_flag_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC1 6MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLD IGSTB	PLLDIG stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag

Enum rcu_int_flag_clear_enum

Table 3-465. Enum rcu_int_flag_clear_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC1 6MSTB_CLR	IRC16M stabilization interrupt flags clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLD IGSTB_CLR	PLLDIG stabilization interrupt flags clear
RCU_INT_FLAG_CKM _CLR	CKM interrupt flags clear

Enum rcu_int_enum

Table 3-466. Enum rcu_int_enum

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLDIGSTB	PLLDIG stabilization interrupt

Enum rcu_osci_type_enum

Table 3-467. Enum rcu_osci_type_enum

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC16M	IRC16M
RCU_IRC32K	IRC32K
RCU_PLLDIG_CK	PLLDIG

Enum rcu_clock_freq_enum

Table 3-468. Enum rcu_clock_freq_enum

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock
CK_USART0	USART0 clock
CK_I2C0	I2C0 clock

rcu_deinit

The description of rcu_deinit is shown as below:

Table 3-469. Function rcu_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU, reset the value of all RCU registers into initial values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* reset RCU */
```

```
rcu_deinit();
```

rcu_irc16m_dfs_to_rf_enable

The description of rcu_irc16m_dfs_to_rf_enable is shown as below:

Table 3-470. Function rcu_irc16m_dfs_to_rf_enable

Function name	rcu_irc16m_dfs_to_rf_enable
Function prototype	void rcu_irc16m_dfs_to_rf_enable(void);
Function descriptions	enable differential signal of IRC16M to RF module
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable differential signal of IRC16M to RF module */
```

```
rcu_irc16m_dfs_to_rf_enable();
```

rcu_irc16m_dfs_to_rf_disable

The description of rcu_irc16m_dfs_to_rf_disable is shown as below:

Table 3-471. Function rcu_irc16m_dfs_to_rf_disable

Function name	rcu_irc16m_dfs_to_rf_disable
Function prototype	void rcu_irc16m_dfs_to_rf_disable(void);
Function descriptions	disable differential signal of IRC16M to RF module
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable differential signal of IRC16M to RF module */
```

```
rcu_irc16m_dfs_to_rf_disable();
```

rcu_periph_clock_enable

The description of rcu_periph_clock_enable is shown as below:

Table 3-472. Function rcu_periph_clock_enable

Function name	rcu_periph_clock_enable;
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph);
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-460. Enum rcu_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

rcu_periph_clock_disable

The description of rcu_periph_clock_disable is shown as below:

Table 3-473. Function rcu_periph_clock_disable

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-460. Enum rcu_periph_enum
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

rcu_fmc_clock_sleep_enable

The description of rcu_fmc_clock_sleep_enable is shown as below:

Table 3-474. Function rcu_periph_clock_sleep_enable

Function name	rcu_fmc_clock_sleep_enable
Function prototype	void rcu_fmc_clock_sleep_enable(void);
Function descriptions	enable the FMC clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_fmc_clock_sleep_enable();
```

rcu_fmc_clock_sleep_disable

The description of rcu_fmc_clock_sleep_disable is shown as below:

Table 3-475. Function rcu_fmc_clock_sleep_disable

Function name	rcu_fmc_clock_sleep_disable
Function prototype	void rcu_fmc_clock_sleep_disable(void);
Function descriptions	disable the FMC clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_fmc_clock_sleep_disable();
```

rcu_periph_reset_enable

The description of rcu_periph_reset_enable is shown as below:

Table 3-476. Function rcu_periph_reset_enable

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-461. Enum rcu_periph_reset_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI reset */
```

```
rcu_periph_reset_enable(RCU_SPIRST);
```

rcu_periph_reset_disable

The description of rcu_periph_reset_disable is shown as below:

Table 3-477. Function rcu_periph_reset_disable

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
Function descriptions	disable the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-461. Enum rcu_periph_reset_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI reset */

rcu_periph_reset_disable(RCU_SPIRST);
```

rcu_bkp_reset_enable

The description of rcu_bkp_reset_enable is shown as below:

Table 3-478. Function rcu_bkp_reset_enable

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the BKP domain */

rcu_bkp_reset_enable();
```

rcu_bkp_reset_disable

The description of rcu_bkp_reset_disable is shown as below:

Table 3-479. Function rcu_bkp_reset_disable

Function name	rcu_bkp_reset_disable
Function prototype	void rcu_bkp_reset_disable(void);
Function descriptions	disable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

rcu_rfppl_cal_enable

The description of rcu_rfppl_cal_enable is shown as below:

Table 3-480. Function rcu_rfppl_cal_enable

Function name	rcu_rfppl_cal_enable
Function prototype	void rcu_rfppl_cal_enable(void);
Function descriptions	enable the RF PLL calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the RF PLL calculation */
```

```
rcu_rfppl_cal_enable ();
```

rcu_rfppl_cal_disable

The description of rcu_rfppl_cal_disable is shown as below:

Table 3-481. Function rcu_rfppl_cal_disable

Function name	rcu_rfppl_cal_disable
Function prototype	void rcu_rfppl_cal_disable(void);
Function descriptions	disable the RF PLL calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the RF PLL calculation */
```

```
rcu_rfpll_cal_disable();
```

rcu_control_unit_powerup

The description of rcu_control_unit_powerup is shown as below:

Table 3-482. Function rcu_control_unit_powerup

Function name	rcu_control_unit_powerup
Function prototype	void rcu_control_unit_powerup (rcu_unit_enum rcu_unit);
Function descriptions	power on the clock
Precondition	-
The called functions	-
Input parameter{in}	
rcu_unit	RCU unit
<i>RCU_UNIT_HXTAL</i>	power on the HXTAL
<i>RCU_UNIT_PLLDIG</i>	power on the PLLDIG
<i>RCU_UNIT_RFPLL</i>	enable the RF PLL calculation
<i>RCU_UNIT_LDOANA</i>	power on LDO analog
<i>RCU_UNIT_LDOCLK</i>	power on the LDO clock
<i>RCU_UNIT_BANDGAP</i>	power on the BandGap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* power on the HXTAL */
rcu_control_unit_powerup(RCU_UNIT_HXTAL);
```

rcu_control_unit_powerdown

The description of rcu_control_unit_powerdown is shown as below:

Table 3-483. Function rcu_control_unit_powerdown

Function name	rcu_control_unit_powerdown
Function prototype	void rcu_control_unit_powerdown (rcu_unit_enum rcu_unit);
Function descriptions	power down the clock
Precondition	-
The called functions	-
Input parameter{in}	
rcu_unit	RCU unit
<i>RCU_UNIT_HXTAL</i>	power down the HXTAL
<i>RCU_UNIT_PLLDIG</i>	power down the PLLDIG
<i>RCU_UNIT_RFPLL</i>	disenable the RF PLL calculation

<i>RCU_UNIT_LDOANA</i>	power down LDO analog
<i>RCU_UNIT_LDOCLK</i>	power down the LDO clock
<i>RCU_UNIT_BANDGAP</i>	power down the BandGap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* power down the HXTAL */
```

```
rcu_control_unit_powerdown(RCU_UNIT_HXTAL);
```

rcu_system_clock_source_config

The description of rcu_system_clock_source_config is shown as below:

Table 3-484. Function rcu_system_clock_source_config

Function name	rcu_system_clock_source_config
Function prototype	void rcu_system_clock_source_config(uint32_t ck_sys);
Function descriptions	configure the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
ck_sys	system clock source select
<i>RCU_CKSYSSRC_IRC16M</i>	select CK_IRC16M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLLDIG</i>	select CK_PLLDIG as the CK_SYS source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

rcu_system_clock_source_get

The description of rcu_system_clock_source_get is shown as below:

Table 3-485. Function rcu_system_clock_source_get

Function name	rcu_system_clock_source_get
Function prototype	uint32_t rcu_system_clock_source_get(void);
Function descriptions	get the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLLDIG

Example:

```
uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();
```

rcu_ahb_clock_config

The description of rcu_ahb_clock_config is shown as below:

Table 3-486. Function rcu_ahb_clock_config

Function name	rcu_ahb_clock_config
Function prototype	void rcu_ahb_clock_config(uint32_t ck_ahb);
Function descriptions	configure the AHB clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_SYS/128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

rcu_apb1_clock_config

The description of rcu_apb1_clock_config is shown as below:

Table 3-487. Function rcu_apb1_clock_config

Function name	rcu_apb1_clock_config
Function prototype	void rcu_apb1_clock_config(uint32_t ck_apb1);
Function descriptions	configure the APB1 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb1	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D</i> <i>IV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV2</i>	select CK_AHB/2 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV4</i>	select CK_AHB/4 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV8</i>	select CK_AHB/8 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV16</i>	select CK_AHB/16 as CK_APB1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

rcu_apb2_clock_config

The description of rcu_apb2_clock_config is shown as below:

Table 3-488. Function rcu_apb2_clock_config

Function name	rcu_apb2_clock_config
Function prototype	void rcu_apb2_clock_config(uint32_t ck_apb2);
Function descriptions	configure the APB2 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb2	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_D</i>	select CK_AHB as CK_APB2

IV1	
RCU_APB2_CK_AHB_D	select CK_AHB/2 as CK_APB2
IV2	
RCU_APB2_CK_AHB_D	select CK_AHB/4 as CK_APB2
IV4	
RCU_APB2_CK_AHB_D	select CK_AHB/8 as CK_APB2
IV8	
RCU_APB2_CK_AHB_D	select CK_AHB/16 as CK_APB2
IV16	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

rcu_ckout0_config

The description of rcu_ckout0_config is shown as below:

Table 3-489. Function rcu_ckout0_config

Function name	rcu_ckout0_config
Function prototype	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
Function descriptions	configure the CK_OUT0 clock source and divider
Precondition	-
The called functions	-
Input parameter{in}	
ckout0_src	CK_OUT0 clock source selection
RCU_CKOUT0SRC_IR	select IRC16M
C16M	
RCU_CKOUT0SRC_LX	select LXTAL
TAL	
RCU_CKOUT0SRC_H	select HXTAL
XTAL	
RCU_CKOUT0SRC_PL	select PLLDIG
LDIG	
RCU_CKOUT0SRC_IR	select IRC32K
C32K	
RCU_CKOUT0SRC_C	select system clock
KSYS	
Input parameter{in}	

ckout0_div	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx</i>	CK_OUT0 is divided by x(x=1,2,3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

rcu_ckout1_config

The description of rcu_ckout1_config is shown as below:

Table 3-490. Function rcu_ckout1_config

Function name	rcu_ckout1_config
Function prototype	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
Function descriptions	configure the CK_OUT1 clock source
Precondition	-
The called functions	-
Input parameter{in}	
ckout1_src	CK_OUT1 clock source selection
<i>RCU_CKOUT1SRC_CKSYS</i>	select system clock
<i>RCU_CKOUT1SRC_IRC16M</i>	select IRC16M
<i>RCU_CKOUT1SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT1SRC_PLLDIG</i>	select PLLDIG
Input parameter{in}	
ckout1_div	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx</i>	CK_OUT1 is divided by x(x=1,2,3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

rcu_plldig_config

The description of rcu_plldig_config is shown as below:

Table 3-491. Function rcu_pll_config

Function name	rcu_plldig_config
Function prototype	void rcu_plldig_config(uint32_t plldig_clk);
Function descriptions	configure the PLLDIG output clock
Precondition	-
The called functions	-
Input parameter{in}	
plldig_clk	PLLDIG output clock selection
<i>RCU_PLLDIG_192M</i>	selected 192Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_240M</i>	selected 240Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_320M</i>	selected 320Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_480M</i>	selected 480Mhz as PLLDIG output frequency
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLDIG output 192Mhz clock frequency */
```

```
rcu_plldig_config(RCU_PLLDIG_192M);
```

rcu_plldigdiv_sys_config

The description of rcu_plldigdiv_sys_config is shown as below:

Table 3-492. Function rcu_plldigdiv_sys_config

Function name	rcu_plldigdiv_sys_config
Function prototype	void rcu_plldigdiv_sys_config(uint32_t plldigdiv_sys);
Function descriptions	configure PLLDIG clock divider factor for system clock
Precondition	-
The called functions	-
Input parameter{in}	
plldigdiv_sys	PLLDIG clock divider factor for system clock
<i>RCU_PLLDIG_SYS_DIVx</i>	PLLDIG clock divided by x for system clock(x=1,2,3,...,64)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PLLDIG clock divider 2 for system clock */
rcu_plldigdiv_sys_config (RCU_PLLDIG_SYS_DIV2);
```

rcu_rtc_clock_config

The description of rcu_rtc_clock_config is shown as below:

Table 3-493. Function rcu_rtc_clock_config

Function name	rcu_rtc_clock_config
Function prototype	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
Function descriptions	configure the RTC clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
rtc_clock_source	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC32K</i>	select CK_IRC32K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_RTCDIV</i>	select CK_HXTAL/RTCDIV as RTC source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

rcu_rtc_div_config

The description of rcu_rtc_div_config is shown as below:

Table 3-494. Function rcu_rtc_div_config

Function name	rcu_rtc_div_config
Function prototype	void rcu_rtc_div_config(uint32_t rtc_div);
Function descriptions	configure the frequency division of RTC clock when HXTAL was selected as its clock source
Precondition	-
The called functions	-
Input parameter{in}	

rtc_div	RTC clock frequency division
<i>RCU_RTC_HXTAL_DIVx</i>	RTCDIV clock select CK_HXTAL/x, x = 1....32
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTCDIV clock select CK_HXTAL/2 */
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV2);
```

rcu_trng_div_config

The description of rcu_trng_div_config is shown as below:

Table 3-495. Function rcu_trng_div_config

Function name	rcu_trng_div_config
Function prototype	void rcu_trng_div_config(uint32_t trng_div);
Function descriptions	configure the frequency division of the TRNG source clock
Precondition	-
The called functions	-
Input parameter{in}	
trng_div	TRNG clock frequency division
<i>RCU_TRNG_DIVx</i>	RTCDIV clock select CK_HXTAL/x, x = 1....32
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TRNG clock divide 2 */
rcu_trng_div_config (RCU_TRNG_DIV2);
```

rcu_i2c0_clock_config

The description of rcu_i2c0_clock_config is shown as below:

Table 3-496. Function rcu_i2c0_clock_config

Function name	rcu_i2c0_clock_config
Function prototype	void rcu_i2c0_clock_config(uint32_t i2c0_clock_source);
Function descriptions	configure the I2C0 clock source selection
Precondition	-

The called functions	-
Input parameter{in}	
i2c0_clock_source	I2C0 clock source selection
<i>RCU_I2C0SRC_CKAPB1</i>	CK_APB1 selected as I2C0 source clock
<i>RCU_I2C0SRC_CKSYS</i>	CK_SYS selected as I2C0 source clock
<i>RCU_I2C0SRC_IRC16M</i>	CK_IRC16M selected as I2C0 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select IRC16M as I2C0 source clock */
rcu_i2c0_clock_config(RCU_I2C0SRC_IRC16M);
```

rcu_usart0_clock_config

The description of rcu_usart0_clock_config is shown as below:

Table 3-497. Function rcu_usart0_clock_config

Function name	rcu_usart0_clock_config
Function prototype	void rcu_usart0_clock_config(uint32_t usart0_clock_source);
Function descriptions	configure the USART0 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
usart0_clock_source	USART0 clock source selection
<i>RCU_USART0SRC_CKAPB1</i>	CK_APB1 selected as USART0 source clock
<i>RCU_USART0SRC_CKSYS</i>	CK_SYS selected as USART0 source clock
<i>RCU_USART0SRC_CK_LXTAL</i>	CK_LXTAL selected as USART0 source clock
<i>RCU_USART0SRC_IRC16M</i>	CK_IRC16M selected as USART0 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select APB1 clock as USART0 clock */
```

```
rcu_usart0_clock_config(RCU_USART0SRC_CKAPB1);
```

rcu_irc16m_div_config

The description of rcu_irc16m_div_config is shown as below:

Table 3-498. Function rcu_irc16m_div_config

Function name	rcu_irc16m_div_config
Function prototype	void rcu_irc16m_div_config(uint32_t irc16m_div);
Function descriptions	configure IRC16M clock divider factor for system clock
Precondition	-
The called functions	-
Input parameter{in}	
irc16m_div	IRC16M clock divider factor for system clock
<i>RCU_IRC16M_DIVx</i>	IRC16M clock divided by x for system clock (x=1,2,3,...,512)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* IRC16M clock divided by 4 for system clock */
```

```
rcu_irc16m_div_config(RCU_IRC16M_DIV4);
```

rcu_timer_clock_prescaler_config

The description of rcu_timer_clock_prescaler_config is shown as below:

Table 3-499. Function rcu_sdio_div_config

Function name	rcu_timer_clock_prescaler_config
Function prototype	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
Function descriptions	configure the TIMER clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_clock_prescaler	TIMER clock selection
<i>RCU_TIMER_PSC_MU_L2</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx
<i>RCU_TIMER_PSC_MU_L4</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER clock source */
```

```
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

rcu_flag_get

The description of rcu_flag_get is shown as below:

Table 3-500. Function rcu_flag_get

Function name	rcu_flag_get
Function prototype	FlagStatus rcu_flag_get(rcu_flag_enum flag);
Function descriptions	get the clock stabilization and peripheral reset flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	the clock stabilization and peripheral reset flags, refer to Table 3-463. Enum rcu_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the clock stabilization flag */
```

```
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

rcu_all_reset_flag_clear

The description of rcu_all_reset_flag_clear is shown as below:

Table 3-501. Function rcu_all_reset_flag_clear

Function name	rcu_all_reset_flag_clear
Function prototype	void rcu_all_reset_flag_clear(void);
Function descriptions	clear all the reset flag
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* clear all the reset flag */
```

```
rcu_all_reset_flag_clear();
```

rcu_interrupt_flag_get

The description of rcu_interrupt_flag_get is shown as below:

Table 3-502. Function rcu_interrupt_flag_get

Function name	rcu_interrupt_flag_get
Function prototype	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
Function descriptions	get the clock stabilization interrupt and ckm flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt and ckm flags, refer to Table 3-463. Enum rcu flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
```

```
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

rcu_interrupt_flag_clear

The description of rcu_interrupt_flag_clear is shown as below:

Table 3-503. Function rcu_interrupt_flag_clear

Function name	rcu_interrupt_flag_clear
Function prototype	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag);
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	clock stabilization and stuck interrupt flags clear, refer to Table 3-465. Enum rcu_int_flag_clear_enum

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

rcu_interrupt_enable

The description of rcu_interrupt_enable is shown as below:

Table 3-504. Function rcu_interrupt_enable

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum interrupt);
Function descriptions	enable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	clock stabilization interrupt, refer to Table 3-466. Enum rcu_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

rcu_interrupt_disable

The description of rcu_interrupt_disable is shown as below:

Table 3-505. Function rcu_interrupt_disable

Function name	rcu_interrupt_disable
Function prototype	void rcu_interrupt_disable(rcu_int_enum interrupt);
Function descriptions	disable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	clock stabilization interrupt, refer to Table 3-466. Enum rcu_int_enum
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

rcu_lxtal_drive_capability_config

The description of rcu_lxtal_drive_capability_config is shown as below:

Table 3-506. Function rcu_lxtal_drive_capability_config

Function name	rcu_lxtal_drive_capability_config
Function prototype	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
lxtal_dricap	drive capability of LXTAL
<i>RCU_LXTALDRI_LOWER_DRIVE</i>	lower driving capability
<i>RCU_LXTALDRI_HIGH_DRIVE</i>	high driving capability
<i>RCU_LXTALDRI_HIGHER_DRIVE</i>	higher driving capability
<i>RCU_LXTALDRI_HIGHEST_DRIVE</i>	highest driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config (RCU_LXTALDRI_LOWER_DRIVE);
```

rcu_osci_stab_wait

The description of rcu_osci_stab_wait is shown as below:

Table 3-507. Function rcu_osci_stab_wait

Function name	rcu_osci_stab_wait
Function prototype	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);

Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-467. Enum rcu_osci_type_enum
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}

```

rcu_osci_on

The description of rcu_osci_on is shown as below:

Table 3-508. Function rcu_osci_on

Function name	rcu_osci_on
Function prototype	void rcu_osci_on(rcu_osci_type_enum osci);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-467. Enum rcu_osci_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osci_on(RCU_HXTAL);

```

rcu_osci_off

The description of rcu_osci_off is shown as below:

Table 3-509. Function rcu_osci_off

Function name	rcu_osci_off
Function prototype	void rcu_osci_off(rcu_osci_type_enum osci);

Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-467. Enum rcu_osci_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

rcu_osci_bypass_mode_enable

The description of rcu_osci_bypass_mode_enable is shown as below:

Table 3-510. Function rcu_osci_bypass_mode_enable

Function name	rcu_osci_bypass_mode_enable
Function prototype	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-467. Enum rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

rcu_osci_bypass_mode_disable

The description of rcu_osci_bypass_mode_disable is shown as below:

Table 3-511. Function rcu_osci_bypass_mode_disable

Function name	rcu_osci_bypass_mode_disable
----------------------	------------------------------

Function prototype	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-467. Enum rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

rcu_rf_hxtal_clock_monitor_enable

The description of rcu_rf_hxtal_clock_monitor_enable is shown as below:

Table 3-512. Function rcu_rf_hxtal_clock_monitor_enable

Function name	rcu_rf_hxtal_clock_monitor_enable
Function prototype	void rcu_rf_hxtal_clock_monitor_enable(void);
Function descriptions	enable the RF HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the RF HXTAL clock monitor */
```

```
rcu_rf_hxtal_clock_monitor_enable();
```

rcu_rf_hxtal_clock_monitor_disable

The description of rcu_rf_hxtal_clock_monitor_disable is shown as below:

Table 3-513. Function rcu_rf_hxtal_clock_monitor_disable

Function name	rcu_rf_hxtal_clock_monitor_disable
Function prototype	void rcu_rf_hxtal_clock_monitor_disable(void);
Function descriptions	disable the RF HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the RF HXTAL clock monitor */
rcu_rf_hxtal_clock_monitor_disable();
```

rcu_irc16m_adjust_value_set

The description of rcu_irc16m_adjust_value_set is shown as below:

Table 3-514. Function rcu_irc16m_adjust_value_set

Function name	rcu_irc16m_adjust_value_set
Function prototype	void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);
Function descriptions	set the IRC16M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc16m_adjval	IRC16M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);
```

rcu_voltage_key_unlock

The description of rcu_voltage_key_unlock is shown as below:

Table 3-515. Function rcu_voltage_key_unlock

Function name	rcu_voltage_key_unlock
Function prototype	void rcu_voltage_key_unlock(void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*unlock the voltage key */
rcu_voltage_key_unlock();
```

rcu_deepsleep_voltage_set

The description of rcu_deepsleep_voltage_set is shown as below:

Table 3-516. Function rcu_deepsleep_voltage_set

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(uint32_t dsvol);
Function descriptions	set the deep-sleep mode voltage value
Precondition	-
The called functions	-
Input parameter{in}	
dsvol	deep sleep mode voltage
RCU_DEEPSLEEP_V_1_1	the core voltage is 1.1V in deep-sleep mode
RCU_DEEPSLEEP_V_1_0	the core voltage is 1.0V in deep-sleep mode
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

rcu_clock_freq_get

The description of rcu_clock_freq_get is shown as below:

Table 3-517. Function rcu_clock_freq_get

Function name	rcu_clock_freq_get
Function prototype	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
Function descriptions	get the system clock, bus clock frequency
Precondition	-
The called functions	-
Input parameter{in}	
clock	Clock frequency, refers to Table 3-468. Enum rcu_clock_freq_enum
Output parameter{out}	
-	-
Return value	
ck_freq	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

3.19. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, two alarms, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.19.1](#), the RTC firmware functions are introduced in chapter [3.19.2](#).

3.19.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

Table 3-518. RTC Registers

Registers	Descriptions
RTC_TIME	time of day register
RTC_DATE	date register
RTC_CTL	control register
RTC_STAT	status register
RTC_PSC	time prescaler register
RTC_WUT	wakeup timer register
RTC_COSC	coarse calibration register

Registers	Descriptions
RTC_ALRM0TD	alarm 0 time and date register
RTC_ALRM1TD	alarm 1 time and date register
RTC_WPK	write protection key register
RTC_SS	sub second register
RTC_SHIFTCTL	shift function control register
RTC_TTS	time of timestamp register
RTC_DTS	date of timestamp register
RTC_SSTS	sub second of timestamp register
RTC_HRFC	high resolution frequency compensation register
RTC_TAMP	tamper register
RTC_ALRM0SS	alarm 0 sub second register
RTC_ALRM1SS	alarm1 sub second register
RTC_BKPx(x = 0, 1, 2, ..., 18, 19)	backup register

3.19.2. Descriptions of Peripheral functions

RTC firmware functions are listed in the table shown as below:

Table 3-519. RTC firmware function

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper

Function name	Function description
rtc_software_bkp_reset	reset the RTC_BKP registers by software
rtc_tamper_without_bkp_seset	tamperx event does not erase the RTC_BKP registers
rtc_output_pin_select	select the RTC output pin
rtc_alarm_output_config	configure RTC alarm output source
rtc_calibration_output_config	configure RTC calibration output source
rtc_hour_adjust	ajust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	ajust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_wakeup_enable	enable RTC wakeup timer
rtc_wakeup_disable	disable RTC wakeup timer
rtc_wakeup_clock_set	set auto wakeup timer clock
rtc_wakeup_timer_set	set auto wakeup timer value
rtc_wakeup_timer_get	get auto wakeup timer value
rtc_smooth_calibration_config	configure RTC smooth calibration
rtc_coarse_calibration_enable	enable RTC coarse calibration
rtc_coarse_calibration_disable	disable RTC coarse calibration
rtc_coarse_calibration_config	configure RTC coarse calibration direction and step
rtc_flag_get	get specified flag
rtc_flag_clear	clear specified flag
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt

Structure rtc_parameter_struct

Table 3-520. Structure rtc_parameter_struct

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value
hour	RTC hour value
minute	RTC minute value: 0x0 - 0x59(BCD format)
second	RTC second value: 0x0 - 0x59(BCD format)
factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
am_pm	RTC AM/PM value
display_format	RTC time notation

Structure rtc_alarm_struct

Table 3-521. Structure rtc_alarm_struct

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value
alarm_hour	RTC alarm hour value
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

Structure rtc_timestamp_struct

Table 3-522. Structure rtc_timestamp_struct

Member name	Function description
timestamp_month	RTC time-stamp month value
timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value
timestamp_hour	RTC time-stamp hour value
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

Structure rtc_tamper_struct

Table 3-523. Structure rtc_tamper_struct

Member name	Function description
tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
tamper_with_timestamp	RTC tamper time-stamp feature

rtc_deinit

The description of rtc_deinit is shown as below:

Table 3-524. Function rtc_deinit

Function name	rtc_deinit
Function prototype	ErrStatus rtc_deinit(void);
Function descriptions	reset most of the RTC registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_deinit();
```

rtc_init

The description of rtc_init is shown as below:

Table 3-525. Function rtc_init

Function name	rtc_init
Function prototype	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	initialize RTC registers
Precondition	-
The called functions	-
Input parameter{in}	
rtc_initpara_struct	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure Table 3-520. Structure rtc_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* initialize RTC registers */
```

```
rtc_parameter_struct rtc_initpara;
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

```

rtc_initpara.factor_asyn = prescaler_a;

rtc_initpara.factor_syn = prescaler_s;

rtc_initpara.year = 0x16;

rtc_initpara.day_of_week = RTC_SATURDAY;

rtc_initpara.month = RTC_APR;

rtc_initpara.date = 0x30;

rtc_initpara.display_format = RTC_24HOUR;

rtc_initpara.am_pm = RTC_AM;

rtc_init(&rtc_initpara);

```

rtc_init_mode_enter

The description of rtc_init_mode_enter is shown as below:

Table 3-526. Function rtc_init_mode_enter

Function name	rtc_init_mode_enter
Function prototype	ErrStatus rtc_init_mode_enter(void);
Function descriptions	enter RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/*enter RTC init mode*/

ErrStatus error_status;

error_status = rtc_init_mode_enter ();

```

rtc_init_mode_exit

The description of rtc_init_mode_exit is shown as below:

Table 3-527. Function rtc_init_mode_exit

Function name	rtc_init_mode_exit
Function prototype	void rtc_init_mode_exit(void);
Function descriptions	exit RTC init mode

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*exit RTC init mode*/
```

```
rtc_init_mode_exit ();
```

rtc_register_sync_wait

The description of rtc_register_sync_wait is shown as below:

Table 3-528. Function rtc_register_sync_wait

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_register_sync_wait ();
```

rtc_current_time_get

The description of rtc_current_time_get is shown as below:

Table 3-529. Function rtc_current_time_get

Function name	rtc_current_time_get
----------------------	----------------------

Function prototype	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	get current time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_initpara_struct	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure Table 3-520. Structure rtc_parameter_struct
Return value	
-	-

Example:

```
/*get current time and date*/

rtc_parameter_struct rtc_initpara_struct;

rtc_current_time_get (&rtc_initpara_struct);
```

rtc_subsecond_get

The description of rtc_subsecond_get is shown as below:

Table 3-530. Function rtc_subsecond_get

Function name	rtc_subsecond_get
Function prototype	uint32_t rtc_subsecond_get(void);
Function descriptions	get current subsecond value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/

uint32_t sub_second = rtc_subsecond_get();
```

rtc_alarm_config

The description of rtc_alarm_config is shown as below:

Table 3-531. Function rtc_alarm_config

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time)
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Input parameter{in}	
rtc_alarm_time	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure Table 3-521. Structure rtc_alarm_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(RTC_ALARM0,&rtc_alarm_time);
```

rtc_alarm_subsecond_config

The description of rtc_alarm_subsecond_config is shown as below:

Table 3-532. Function rtc_alarm_subsecond_config

Function name	rtc_alarm_subsecond_config
Function prototype	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
Function descriptions	configure subsecond of RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Input parameter{in}	
mask_subsecond	alarm subsecond mask
<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration

<i>RTC_MASKSSC_1_14</i>	mask RTC_ALARM0SS_SSC[14:1], and RTC_ALARM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALARM0SS_SSC[14:2], and RTC_ALARM0SS_SSC[1:0] is to be compared
<i>RTC_MASKSSC_3_14</i>	mask RTC_ALARM0SS_SSC[14:3], and RTC_ALARM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALARM0SS_SSC[14:4], and RTC_ALARM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALARM0SS_SSC[14:5], and RTC_ALARM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALARM0SS_SSC[14:6], and RTC_ALARM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALARM0SS_SSC[14:7], and RTC_ALARM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALARM0SS_SSC[14:8], and RTC_ALARM0SS_SSC[7:0] is to be compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALARM0SS_SSC[14:9], and RTC_ALARM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i>	mask RTC_ALARM0SS_SSC[14:10], and RTC_ALARM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALARM0SS_SSC[14:11], and RTC_ALARM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALARM0SS_SSC[14:12], and RTC_ALARM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALARM0SS_SSC[14:13], and RTC_ALARM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALARM0SS_SSC[14], and RTC_ALARM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALARM0SS_SSC[14:0] is to be compared
Input parameter{in}	
subsecond	alarm subsecond value(0x000 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure subsecond of RTC alarm0*/
```

```
rtc_subsecond_config (RTC_ALARM0,RTC_MASKSSC_9_14, 0x7FFF);
```

rtc_alarm_get

The description of rtc_alarm_get is shown as below:

Table 3-533. Function rtc_alarm_get

Function name	rtc_alarm_get
Function prototype	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
Function descriptions	get RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Output parameter{out}	
rtc_alarm_time	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure Table 3-521. Structure rtc_alarm_struct
Return value	
-	-

Example:

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_get(RTC_ALARM0,&rtc_alarm_time);
```

rtc_alarm_subsecond_get

The description of rtc_alarm_subsecond_get is shown as below:

Table 3-534. Function rtc_alarm_subsecond_get

Function name	rtc_alarm_subsecond_get
Function prototype	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm)
Function descriptions	get RTC alarm subsecond
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Output parameter{out}	
-	-
Return value	
uint32_t	RTC alarm subsecond value(0x0-0x7FFF)

Example:

```
/*get RTC alarm0 subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

rtc_alarm_enable

The description of rtc_alarm_enable is shown as below:

Table 3-535. Function rtc_alarm_enable

Function name	rtc_alarm_enable
Function prototype	void rtc_alarm_enable(uint8_t rtc_alarm)
Function descriptions	enable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC alarm0*/
```

```
rtc_alarm_enable(RTC_ALARM0);
```

rtc_alarm_disable

The description of rtc_alarm_disable is shown as below:

Table 3-536. Function rtc_alarm_disable

Function name	rtc_alarm_disable
Function prototype	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm)
Function descriptions	disable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Output parameter{out}	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*disable RTC alarm0*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_alarm_disable(RTC_ALARM0);
```

rtc_timestamp_enable

The description of rtc_timestamp_enable is shown as below:

Table 3-537. Function rtc_timestamp_enable

Function name	rtc_timestamp_enable
Function prototype	void rtc_timestamp_enable(uint32_t edge);
Function descriptions	enable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
edge	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

rtc_timestamp_disable

The description of rtc_timestamp_disable is shown as below:

Table 3-538. Function rtc_timestamp_disable

Function name	rtc_timestamp_disable
Function prototype	void rtc_timestamp_disable(void);
Function descriptions	disable RTC time-stamp
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC time-stamp */
```

```
rtc_timestamp_disable ();
```

rtc_timestamp_get

The description of rtc_timestamp_get is shown as below:

Table 3-539. Function rtc_timestamp_get

Function name	rtc_timestamp_get
Function prototype	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
Function descriptions	get RTC timestamp time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_timestamp	Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure Table 3-523. Structure rtc_tamper_struct
Return value	
-	-

Example:

```
/* get RTC timestamp time and date */
```

```
rtc_timestamp_struct rtc_timestamp;
```

```
rtc_timestamp_get(& rtc_timestamp);
```

rtc_timestamp_subsecond_get

The description of rtc_timestamp_subsecond_get is shown as below:

Table 3-540. Function rtc_timestamp_subsecond_get

Function name	rtc_timestamp_subsecond_get
Function prototype	uint32_t rtc_timestamp_subsecond_get(void);
Function descriptions	get RTC time-stamp subsecond

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

rtc_tamper_enable

The description of rtc_tamper_enable is shown as below:

Table 3-541. Function rtc_tamper_enable

Function name	rtc_tamper_enable
Function prototype	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
Function descriptions	enable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
rtc_tamper	pointer to a rtc_tamper_struct structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure Table 3-523. Structure rtc_tamper_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC tamper */
```

```
rtc_tamper_struct rtc_tamper
```

```
rtc_tamper_enable(& rtc_tamper);
```

rtc_tamper_disable

The description of rtc_tamper_disable is shown as below:

Table 3-542. Function rtc_tamper_disable

Function name	rtc_tamper_disable
---------------	--------------------

Function prototype	void rtc_tamper_disable(uint32_t source);
Function descriptions	disable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
source	specify which tamper source to be disabled
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

rtc_software_bkp_reset

The description of rtc_software_bkp_reset is shown as below:

Table 3-543. Function rtc_software_bkp_reset

Function name	rtc_software_bkp_reset
Function prototype	void rtc_software_bkp_reset(void)
Function descriptions	reset the RTC_BKP registers by software
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the RTC_BKP registers by software */
rtc_software_bkp_reset ();
```

rtc_tamper_without_bkp_seset

The description of rtc_tamper_without_bkp_seset is shown as below:

Table 3-544. Function rtc_tamper_without_bkp_seset

Function name	rtc_tamper_without_bkp_seset
Function prototype	void rtc_tamper_without_bkp_seset(uint32_t ne_source)
Function descriptions	tamperx event does not erase the RTC_BKP registers
Precondition	-
The called functions	-
Input parameter{in}	
ne_source	specify which tamper source will not trigger the RTC_BKP registers reset
<i>RTC_TAMPXNOER_NONE</i>	both tamper0 and tamper1 event will trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP0</i>	tamper0 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP1</i>	tamper1 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP0_TP1</i>	neither tamper0 nor tamper1 event will trigger RTC_BKP registers reset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set tamper0 will not trigger RTC_BKP registers */
rtc_tamper_without_bkp_seset(RTC_TAMPXNOER_TP0);
```

rtc_output_pin_select

The description of rtc_output_pin_select is shown as below:

Table 3-545. Function rtc_output_pin_select

Function name	rtc_output_pin_select
Function prototype	void rtc_output_pin_select(uint32_t pin);
Function descriptions	select the RTC output pin
Precondition	-
The called functions	-
Input parameter{in}	
pad	specify the rtc output pad is PC15 or not
<i>RTC_OUT_PC15</i>	the rtc output pad is PC15
<i>RTC_OUT_PA3_PA8</i>	the rtc output pad is PA3 or PA8 according to the AFIO configuration
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select the rtc output pad is PC15 */
rtc_output_pad_select(RTC_OUT_PC15);
```

rtc_alarm_output_config

The description of rtc_alarm_output_config is shown as below:

Table 3-546. Function rtc_alarm_output_config

Function name	rtc_alarm_output_config
Function prototype	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
Function descriptions	configure rtc alarm output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
RTC_ALARM0_HIGH	when the alarm0 flag is set, the output pin is high
RTC_ALARM0_LOW	when the alarm0 flag is set, the output pin is low
RTC_ALARM1_HIGH	when the alarm1 flag is set, the output pin is high
RTC_ALARM1_LOW	when the alarm1 flag is set, the output pin is low
RTC_WAKEUP_HIGH	when the wakeup flag is set, the output pin is high
RTC_WAKEUP_LOW	when the wakeup flag is set, the output pin is low
Input parameter{in}	
mode	specify the output pin mode when output alarm signal or auto wakeup signal
RTC_ALARM_OUTPUT_OD	open drain mode
RTC_ALARM_OUTPUT_PP	push pull mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc alternate output source */
rtc_alter_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

rtc_calibration_output_config

The description of rtc_calibration_output_config is shown as below:

Table 3-547. Function rtc_calibration_output_config

Function name	rtc_calibration_output_config
---------------	-------------------------------

Function prototype	void rtc_calibration_output_config(uint32_t source);
Function descriptions	configure rtc calibration output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
<i>RTC_CALIBRATION_512HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc calibration output source */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

rtc_hour_adjust

The description of rtc_hour_adjust is shown as below:

Table 3-548. Function rtc_hour_adjust

Function name	rtc_hour_adjust
Function prototype	void rtc_hour_adjust(uint32_t operation);
Function descriptions	adjust the daylight saving time by adding or subtracting one hour from the current time
Precondition	-
The called functions	-
Input parameter{in}	
operation	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

rtc_second_adjust

The description of rtc_second_adjust is shown as below:

Table 3-549. Function rtc_second_adjust

Function name	rtc_second_adjust
Function prototype	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
Function descriptions	adjust RTC second or subsecond value of current time
Precondition	-
The called functions	-
Input parameter{in}	
add	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S ET</i>	add 1s to current time
Input parameter{in}	
minus	number of subsecond to minus from current time(0x0 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */
ErrStatus error_status;

error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

rtc_bypass_shadow_enable

The description of rtc_bypass_shadow_enable is shown as below:

Table 3-550. Function rtc_bypass_shadow_enable

Function name	rtc_bypass_shadow_enable
Function prototype	void rtc_bypass_shadow_enable(void);
Function descriptions	enable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable RTC bypass shadow registers function */
```

```
rtc_bypass_shadow_enable();
```

rtc_bypass_shadow_disable

The description of rtc_bypass_shadow_disable is shown as below:

Table 3-551. Function rtc_bypass_shadow_disable

Function name	rtc_bypass_shadow_disable
Function prototype	void rtc_bypass_shadow_disable (void);
Function descriptions	disable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC bypass shadow registers function */
```

```
rtc_bypass_shadow_disable ();
```

rtc_refclock_detection_enable

The description of rtc_refclock_detection_enable is shown as below:

Table 3-552. Function rtc_refclock_detection_enable

Function name	rtc_refclock_detection_enable
Function prototype	ErrStatus rtc_refclock_detection_enable(void);
Function descriptions	enable RTC reference clock detection function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_enable();
```

rtc_refclock_detection_disable

The description of rtc_refclock_detection_disable is shown as below:

Table 3-553. Function rtc_refclock_detection_disable

Function name	rtc_refclock_detection_disable
Function prototype	ErrStatus rtc_refclock_detection_disable(void);
Function descriptions	disable RTC reference clock detection function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_disable ();
```

rtc_wakeup_enable

The description of rtc_refclock_detection_disable is shown as below:

Table 3-554. Function rtc_wakeup_enable

Function name	rtc_wakeup_enable
Function prototype	void rtc_wakeup_enable(void);
Function descriptions	enable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable RTC auto wakeup function */
```

```
rtc_wakeup_enable();
```

rtc_wakeup_disable

The description of rtc_wakeup_disable is shown as below:

Table 3-555. Function rtc_wakeup_disable

Function name	rtc_wakeup_disable
Function prototype	ErrStatus rtc_wakeup_disable(void);
Function descriptions	disable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
```

```
ErrStatus error_status;
```

```
error_status = rtc_wakeup_disable ();
```

rtc_wakeup_clock_set

The description of rtc_wakeup_clock_set is shown as below:

Table 3-556. Function rtc_wakeup_clock_set

Function name	rtc_wakeup_clock_set
Function prototype	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
Function descriptions	set RTC auto wakeup timer clock
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_clock	wakeup timer clock is RTC clock divided factor
WAKEUP_RTCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV	RTC auto wakeup timer clock is RTC clock divided by 8

8	
WAKEUP_RTCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCK_DIV 2	RTC auto wakeup timer clock is RTC clock divided by 2
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2 EXP16	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
```

```
ErrStatus error_status;
```

```
error_status = rtc_wakeup_clock_set(WAKEUP_RTCK_DIV8);
```

rtc_wakeup_timer_set

The description of rtc_wakeup_timer_set is shown as below:

Table 3-557. Function rtc_wakeup_timer_set

Function name	rtc_wakeup_timer_set
Function prototype	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_timer	wakeup timer value
uint16_t	0x0000-0xffff
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
```

```
ErrStatus error_status;
```

```
error_status = rtc_wakeup_timer_set(0XFFEE);
```

rtc_wakeup_timer_get

The description of rtc_wakeup_timer_set is shown as below:

Table 3-558. Function rtc_wakeup_timer_get

Function name	rtc_wakeup_timer_get
Function prototype	uint16_t rtc_wakeup_timer_get(void);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0xFFFF

Example:

```
/* get wakeup timer value */
uint32_t wakeup_time;

wakeup_time = rtc_wakeup_timer();
```

rtc_smooth_calibration_config

The description of rtc_smooth_calibration_config is shown as below:

Table 3-559. Function rtc_smooth_calibration_config

Function name	rtc_smooth_calibration_config
Function prototype	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	configure RTC smooth calibration
Precondition	-
The called functions	-
Input parameter{in}	
window	select calibration window
RTC_CALIBRATION_WINDOW_32S	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_16S	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_8S	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
Input parameter{in}	
plus	add RTC clock or not

<i>RTC_CALIBRATION_PLUS_SET</i>	add one RTC clock every 2048 rtc clock
<i>RTC_CALIBRATION_PLUS_RESET</i>	no effect
Input parameter{in}	
minus	the RTC clock to minus during the calibration window(0x0 - 0xFF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure RTC smooth calibration */
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S, RTC_CALIBRATION_PLUS_SET, 0x10);
```

rtc_coarse_calibration_enable

The description of `rtc_coarse_calibration_enable` is shown as below:

Table 3-560. Function `rtc_coarse_calibration_enable`

Function name	<code>rtc_coarse_calibration_enable</code>
Function prototype	<code>ErrStatus rtc_coarse_calibration_enable(void);</code>
Function descriptions	enable RTC coarse calibration
Precondition	-
The called functions	<code>rtc_init_mode_enter/rtc_init_mode_exit</code>
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC coarse calibration */
```

```
rtc_coarse_calibration_enable ();
```

rtc_coarse_calibration_disable

The description of `rtc_coarse_calibration_disable` is shown as below:

Table 3-561. Function rtc_coarse_calibration_disable

Function name	rtc_coarse_calibration_disable
Function prototype	ErrStatus rtc_coarse_calibration_disable(void);
Function descriptions	disable RTC coarse calibration
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* rtc_coarse_calibration_disable */

ErrStatus error_status;

error_status = rtc_coarse_calibration_disable ();
```

rtc_coarse_calibration_config

The description of rtc_coarse_calibration_config is shown as below:

Table 3-562. Function rtc_coarse_calibration_config

Function name	rtc_coarse_calibration_config
Function prototype	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
Function descriptions	config coarse calibration direction and step
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
direction	coarse calibration direction
<i>CALIB_INCREASE</i>	Increase calendar update frequency
<i>CALIB_DECREASE</i>	Decrease calendar update frequency
Input parameter{in}	
step	coarse calibration step
<i>0x00-0x1F</i>	COSD=0:
	0x00: +0PPM
	0x01: +4PPM(approximate value)
	0x02: +8PPM (approximate value)

	0x1F: +126PPM (approximate value)
	COSD=1:
	0x00: -0PPM
	0x01: -2PPM(approximate value)

	0x02: -4PPM (approximate value) 0x1F: -63PPM (approximate value)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config coarse calibration direction and step */
```

```
ErrStatus error_status;
```

```
error_status = rtc_coarse_calibration_config (INCREASE, 0x01);
```

rtc_flag_get

The description of rtc_flag_get is shown as below:

Table 3-563. Function rtc_flag_get

Function name	rtc_flag_get
Function prototype	FlagStatus rtc_flag_get(uint32_t flag);
Function descriptions	check specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to check
<i>RTC_FLAG_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm0 event flag
<i>RTC_FLAG_ALARM1</i>	alarm1event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_INIT</i>	initialization state flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
<i>RTC_FLAG_YCM</i>	year configuration mark status flag
<i>RTC_FLAG_SOP</i>	shift function operation pending flag
<i>RTC_FLAG_ALARM0W</i>	alarm0 configuration can be written flag
<i>RTC_FLAG_ALARM1W</i>	alarm1 configuration can be written flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be written flag
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus flag_status;
```

```
flag_status = rtc_flag_get(RTC_FLAG_TS);
```

rtc_flag_clear

The description of rtc_flag_clear is shown as below:

Table 3-564. Function rtc_flag_clear

Function name	rtc_flag_clear
Function prototype	void rtc_flag_clear(uint32_t flag);
Function descriptions	clear specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to clear
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	alarm1 occurs flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear time-stamp event flag */
```

```
rtc_flag_clear (RTC_FLAG_TS);
```

rtc_interrupt_enable

The description of rtc_interrupt_enable is shown as below:

Table 3-565. Function rtc_interrupt_enable

Function name	rtc_interrupt_enable
---------------	----------------------

Function prototype	void rtc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable specified RTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_ALARM0);
```

rtc_interrupt_disable

The description of rtc_interrupt_disable is shown as below:

Table 3-566. Function rtc_interrupt_disable

Function name	rtc_interrupt_disable
Function prototype	void rtc_interrupt_disable(uint32_t interrupt);
Function descriptions	disble specified RTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disble specified RTC interrupt */
rtc_interrupt_disable(RTC_INT_ALARM0);
```

3.20. SPI

The SPI module can communicate with external devices using the SPI protocol. The SPI registers are listed in chapter [3.20.1](#), the SPI firmware functions are introduced in chapter [3.20.2](#).

3.20.1. Descriptions of Peripheral registers

SPI registers are listed in the table shown as below:

Table 3-567. SPI registers

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register

3.20.2. Descriptions of Peripheral functions

SPI firmware functions are listed in the table shown as below:

Table 3-568. SPI firmware function

Function name	Function description
spi_deinit	reset SPI peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_data_frame_format_config	configure SPI data frame format
spi_data_transmit	SPI transmit data
spi_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_format_error_clear	clear TI mode format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial

Function name	Function description
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_interrupt_enable	enable SPI interrupt
spi_interrupt_disable	disable SPI interrupt
spi_interrupt_flag_get	get SPI interrupt status
spi_flag_get	get SPI flag status

Structure spi_parameter_struct

Table 3-569. Structure spi_parameter_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_xBIT, x=8/16)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

spi_deinit

The description of spi_deinit is shown as below:

Table 3-570. Function spi_deinit

Function name	spi_deinit
Function prototype	void spi_deinit(void);
Function descriptions	reset SPI peripheral

Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SPI */
spi_deinit();
```

spi_struct_para_init

The description of spi_struct_para_init is shown as below:

Table 3-571. Function spi_struct_para_init

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	Initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
spi_struct	SPI init parameter struct, the structure members can refer to Table 3-569. Structure spi_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

spi_init

The description of spi_init is shown as below:

Table 3-572. Function spi_init

Function name	spi_init
Function prototype	void spi_init(spi_parameter_struct* spi_struct);

Function descriptions	initialize SPI peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_struct	SPI parameter initialization struct, the structure members can refer to members of the structure Table 3-569. Structure spi_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize SPI */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CLK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI, &spi_init_struct);

```

spi_enable

The description of spi_enable is shown as below:

Table 3-573. Function spi_enable

Function name	spi_enable
Function prototype	void spi_enable(void);
Function descriptions	enable SPI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI */

spi_enable();
```

spi_disable

The description of spi_disable is shown as below:

Table 3-574. Function spi_disable

Function name	spi_disable
Function prototype	void spi_disable(void);
Function descriptions	disable SPI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI */

spi_disable();
```

spi_nss_output_enable

The description of spi_nss_output_enable is shown as below:

Table 3-575. Function spi_nss_output_enable

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(void);
Function descriptions	enable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI NSS output */
```

```
spi_nss_output_enable();
```

spi_nss_output_disable

The description of spi_nss_output_disable is shown as below:

Table 3-576. Function spi_nss_output_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(void);
Function descriptions	disable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI NSS output */
```

```
spi_nss_output_disable();
```

spi_nss_internal_high

The description of spi_nss_internal_high is shown as below:

Table 3-577. Function spi_nss_internal_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(void);
Function descriptions	SPI NSS pin high level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high();
```

spi_nss_internal_low

The description of spi_nss_internal_low is shown as below:

Table 3-578. Function spi_nss_internal_low

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(void);
Function descriptions	SPI NSS pin low level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low();
```

spi_dma_enable

The description of spi_dma_enable is shown as below:

Table 3-579. Function spi_dma_enable

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI transmit data DMA function */
```

```
spi_dma_enable(SPI_DMA_TRANSMIT);
```

spi_dma_disable

The description of spi_dma_disable is shown as below:

Table 3-580. Function spi_dma_disable

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI transmit data DMA function */
spi_dma_disable(SPI_DMA_TRANSMIT);
```

spi_data_frame_format_config

The description of spi_data_frame_format_config is shown as below:

Table 3-581. Function spi_data_frame_format_config

Function name	spi_data_frame_format_config
Function prototype	void spi_data_frame_format_config(uint16_t frame_format);
Function descriptions	configure SPI data frame format
Precondition	-
The called functions	-
Input parameter{in}	
frame_format	SPI frame size
<i>SPI_FRAME_SIZE_xBIT</i>	SPI frame size is x bits,x=8/16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI data frame format size is 16 bits */
```

```
spi_data_frame_format_config(SPI_FRAME_SIZE_16BIT);
```

spi_data_transmit

The description of spi_data_transmit is shown as below:

Table 3-582. Function spi_data_transmit

Function name	spi_data_transmit
Function prototype	void spi_data_transmit(uint16_t data);
Function descriptions	SPI transmit data
Precondition	-
The called functions	-
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI transmit data */
```

```
spi_data_transmit(spi_send_array[send_n]);
```

spi_data_receive

The description of spi_data_receive is shown as below:

Table 3-583. Function spi_data_receive

Function name	spi_data_receive
Function prototype	uint16_t spi_data_receive(void);
Function descriptions	SPI receive data
Precondition	-
The called functions	-
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI receive data */
```

```
spi_receive_array[receive_n] = spi_data_receive();
```

spi_bidirectional_transfer_config

The description of spi_bidirectional_transfer_config is shown as below:

Table 3-584. Function spi_bidirectional_transfer_config

Function name	spi_bidirectional_transfer_config
Function prototype	void spi_bidirectional_transfer_config(uint32_t transfer_direction);
Function descriptions	configure SPI bidirectional transfer direction
Precondition	-
The called functions	-
Input parameter{in}	
transfer_direction	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI_BIDIRECTIONAL_TRANSMIT);
```

spi_format_error_clear

The description of spi_format_error_clear is shown as below:

Table 3-585. Function spi_format_error_clear

Function name	spi_format_error_clear
Function prototype	void spi_format_error_clear(void);
Function descriptions	clear SPI format error flag status
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI format error flag status */
```

```
spi_format_error_clear();
```

spi_crc_polynomial_set

The description of spi_crc_polynomial_set is shown as below:

Table 3-586. Function spi_crc_polynomial_set

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint16_t crc_poly);
Function descriptions	set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI CRC polynomial */
spi_crc_polynomial_set(CRC_VALUE);
```

spi_crc_polynomial_get

The description of spi_crc_polynomial_get is shown as below:

Table 3-587. Function spi_crc_polynomial_get

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(void);
Function descriptions	get SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	16 bit CRC polynomial value

Example:

```
/* get SPI CRC polynomial */
uint16_t crc_val;
```



```
crc_val = spi_crc_polynomial_get();
```

spi_crc_on

The description of spi_crc_on is shown as below:

Table 3-588. Function spi_crc_on

Function name	spi_crc_on
Function prototype	void spi_crc_on(void);
Function descriptions	turn on CRC function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on SPI CRC function */
```

```
spi_crc_on();
```

spi_crc_off

The description of spi_crc_off is shown as below:

Table 3-589. Function spi_crc_off

Function name	spi_crc_off
Function prototype	void spi_crc_off(void);
Function descriptions	turn off CRC function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off SPI CRC function */
```

```
spi_crc_off();
```

spi_crc_next

The description of spi_crc_next is shown as below:

Table 3-590. Function spi_crc_next

Function name	spi_crc_next
Function prototype	void spi_crc_next(void);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI next data is CRC value */
```

```
spi_crc_next();
```

spi_crc_get

The description of spi_crc_get is shown as below:

Table 3-591. Function spi_crc_get

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint8_t crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
<i>crc</i>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value

Example:

```
/* get SPI CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI_CRC_TX);
```

spi_crc_error_clear

The description of spi_crc_error_clear is shown as below:

Table 3-592. Function spi_crc_error_clear

Function name	spi_crc_error_clear
Function prototype	void spi_crc_error_clear(void);
Function descriptions	clear SPI CRC error flag status
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI CRC error flag status */
```

```
spi_crc_error_clear();
```

spi_ti_mode_enable

The description of spi_ti_mode_enable is shown as below:

Table 3-593. Function spi_ti_mode_enable

Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(void);
Function descriptions	enable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI TI mode */
```

```
spi_ti_mode_enable();
```

spi_ti_mode_disable

The description of spi_ti_mode_disable is shown as below:

Table 3-594. Function spi_ti_mode_disable

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable(void);
Function descriptions	disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI TI mode */
spi_ti_mode_disable();
```

spi_interrupt_enable

The description of spi_interrupt_enable is shown as below:

Table 3-595. Function spi_interrupt_enable

Function name	spi_interrupt_enable
Function prototype	void spi_interrupt_enable(uint8_t interrupt);
Function descriptions	enable SPI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	SPI interrupt
<i>SPI_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_enable(SPI_INT_TBE);
```

spi_interrupt_disable

The description of spi_interrupt_disable is shown as below:

Table 3-596. Function spi_interrupt_disable

Function name	spi_interrupt_disable
Function prototype	void spi_interrupt_disable(uint8_t interrupt);
Function descriptions	disable SPI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	SPI interrupt
<i>SPI_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_disable(SPI_INT_TBE);
```

spi_interrupt_flag_get

The description of spi_interrupt_flag_get is shown as below:

Table 3-597. Function spi_interrupt_flag_get

Function name	spi_interrupt_flag_get
Function prototype	FlagStatus spi_interrupt_flag_get(uint8_t interrupt);
Function descriptions	get SPI interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	SPI interrupt
<i>SPI_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF</i>	config error interrupt

<i>ERR</i>	
<i>SPI_INT_FLAG_CRCE</i>	CRC error interrupt
<i>RR</i>	
<i>SPI_INT_FLAG_FERR</i>	format error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get SPI transmit buffer empty interrupt status */
if(RESET != spi_interrupt_flag_get(SPI_INT_FLAG_TBE)){
    while(RESET == spi_flag_get(SPI_FLAG_TBE));
    spi_data_transmit(spi_send_array[send_n++]);
}

```

spi_flag_get

The description of spi_flag_get is shown as below:

Table 3-598. Function spi_flag_get

Function name	spi_flag_get
Function prototype	FlagStatus spi_flag_get(uint8_t interrupt);
Function descriptions	get SPI flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	SPI flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	SPI format error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get SPI transmit buffer empty flag status */

```

```
while(RESET == spi_flag_get(SPI_FLAG_TBE));

spi_data_transmit(spi_send_array[send_n++]);
```

3.21. SYSCFG

The SYSCFG registers are listed in chapter [3.21.1](#), the SYSCFG firmware functions are introduced in chapter [3.21.2](#).

3.21.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

Table 3-599. SYSCFG Registers

Registers	Descriptions
SYSCFG_CFG0	system configuration register 0
SYSCFG_EXTISS0	EXTI sources selection register 0
SYSCFG_EXTISS1	EXTI sources selection register 1
SYSCFG_EXTISS2	EXTI sources selection register 2
SYSCFG_EXTISS3	EXTI sources selection register 3
SYSCFG_CPSCTL	I/O compensation control register
SYSCFG_CFG1	system configuration register 1
SYSCFG_SCFG	SYSCFG shared SRAM configuration register
SYSCFG_TIMERxCFG	TIMER trigger selection register(x = 0..2)

3.21.2. Descriptions of Peripheral functions

SYSCFG firmware functions are listed in the table shown as below:

Table 3-600. SYSCFG firmware function

Function name	Function description
syscfg_deinit	reset the SYSCFG registers
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_io_compensation_enable	enable I/O compensation cell
syscfg_io_compensation_disable	disable I/O compensation cell
syscfg_lock_config	connect TIMER0/15/16 break input to the selected parameter
syscfg_io_compensation_ready_flag_get	get the compensation ready flag
syscfg_sram_ownership_config	configure the ownership of the shared 32k SRAM
syscfg_boot_mode_get	get code start mode

Enum syscfg_code_start_enum

Table 3-601. Enum syscfg_code_start_enum

enum name	Function description
FLASH_START	flash start mode
SRAM_START	sram start mode
SYSTEM_START	system start mode

syscfg_deinit

The description of syscfg_deinit is shown as below:

Table 3-602. Function syscfg_deinit

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	reset the SYSCFG registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SYSCFG registers */
syscfg_deinit();
```

syscfg_exti_line_config

The description of syscfg_exti_line_config is shown as below:

Table 3-603. Function syscfg_exti_line_config

Function name	syscfg_exti_line_config
Function prototype	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
Function descriptions	configure the GPIO pin as EXTI Line
Precondition	-
The called functions	-
Input parameter{in}	
exti_port	specify the GPIO port used in EXTI
EXTI_SOURCE_GPIOx	x=A,B,C
exti_pin	specify the EXTI line
EXTI_SOURCE_PINx	x=0..15(GPIOA), x=0..4,11,12,13,15(GPIOB) , x=8,13,14,15 (GPIOC)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
```

```
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

syscfg_io_compensation_enable

The description of syscfg_io_compensation_enable is shown as below:

Table 3-604. Function syscfg_io_compensation_enable

Function name	syscfg_io_compensation_enable
Function prototype	void syscfg_io_compensation_enable(void);
Function descriptions	enable I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I/O compensation cell */
```

```
syscfg_io_compensation_enable();
```

syscfg_io_compensation_disable

The description of syscfg_io_compensation_disable is shown as below:

Table 3-605. Function syscfg_io_compensation_disable

Function name	syscfg_io_compensation_disable
Function prototype	void syscfg_io_compensation_disable(void);
Function descriptions	disable I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable I/O compensation cell */

syscfg_io_compensation_disable();
```

syscfg_lock_config

The description of syscfg_lock_config is shown as below:

Table 3-606. Function syscfg_lock_config

Function name	syscfg_lock_config
Function prototype	void syscfg_lock_config(void);
Function descriptions	connect TIMER0/15/16 break input to the selected parameter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* connect TIMER0/15/16 break input to the selected parameter */

syscfg_lock_config();
```

syscfg_io_compensation_ready_flag_get

The description of syscfg_io_compensation_ready_flag_get is shown as below:

Table 3-607. Function syscfg_io_compensation_ready_flag_get

Function name	syscfg_io_compensation_ready_flag_get
Function prototype	FlagStatus syscfg_io_compensation_ready_flag_get(void);
Function descriptions	get the compensation ready flag
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
FlagStatus	RESET / SET

Example:

```
/* get the compensation ready flag */
while(RESET != syscfg_io_compensation_ready_flag_get()){
}
```

syscfg_sram_ownership_config

The description of syscfg_sram_ownership_config is shown as below:

Table 3-608. Function syscfg_sram_ownership_config

Function name	syscfg_sram_ownership_config
Function prototype	void syscfg_sram_ownership_config(uint32_t sram_owner);
Function descriptions	configure the ownership of the shared 32k SRAM
Precondition	-
The called functions	-
Input parameter{in}	
sram_owner	the ownership of the shared SRAM
SRAMEN_WIRELESS	wireless
SRAMEN_CORE	core
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ownership */
syscfg_sram_ownership_config(SRAMEN_WIRELESS);
```

syscfg_boot_mode_get

The description of syscfg_boot_mode_get is shown as below:

Table 3-609. Function syscfg_boot_mode_get

Function name	syscfg_boot_mode_get
Function prototype	syscfg_code_start_enum syscfg_boot_mode_get(void);
Function descriptions	get code start mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
syscfg_code_start_enum	code start mode
<i>FLASH_START</i>	flash start mode
<i>SRAM_START</i>	sram start mode
<i>SYSTEM_START</i>	system start mode

Example:

```
/* get code start mode */
if(FLASH_START==syscfg_boot_mode_get()){
}
```

3.22. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMERx, x=0), general level0 timer (TIMERx, x=1, 2), general level4 timer (TIMERx, x=15, 16), basic timer (TIMERx, x=5). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.22.1](#), the TIMER firmware functions are introduced in chapter [3.22.2](#).

3.22.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

Table 3-610. TIMERx Registers

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register

Registers	Descriptions
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_IRMP	TIMER channel input remap register
TIMER_CFG	Configuration register

3.22.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-611. TIMEx firmware function

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer

Function name	Function description
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_channel_input_remap_config	configure TIMER channel input remap
timer_input_trigger_source_select	select TIMER input trigger source

Function name	Function description
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

Structure timer_parameter_struct

Table 3-612. Structure timer_parameter_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

Structure timer_break_parameter_struct

Table 3-613. Structure timer_break_parameter_struct

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
idleoffstate	idle mode off-state (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time (0~255)
breakpolarity	break polarity (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	complementary register protect control (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	break enable (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

Structure timer_oc_parameter_struct

Table 3-614. Structure timer_oc_parameter_struct

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

Structure timer_ic_parameter_struct

Table 3-615. Structure timer_ic_parameter_struct

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control (0~15)

timer_deinit

The description of timer_deinit is shown as below:

Table 3-616. Function timer_deinit

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (x=0, 1, 2, 5, 15, 16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit(TIMER0);
```

timer_struct_para_init

The description of timer_struct_para_init is shown as below:

Table 3-617. Function timer_struct_para_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-612. Structure timer_parameter_struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(&timer_initpara);
```

timer_init

The description of timer_init is shown as below:

Table 3-618. Function timer_init

Function name	timer_init
Function prototype	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
Function descriptions	initialize TIMER counter
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-612. Structure timer_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);
```

timer_enable

The description of timer_enable is shown as below:

Table 3-619. Function timer_enable

Function name	timer_enable
Function prototype	void timer_enable(uint32_t timer_periph);
Function descriptions	enable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2,5,15,16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 */
timer_enable(TIMER0);
```

timer_disable

The description of timer_disable is shown as below:

Table 3-620. Function timer_disable

Function name	timer_disable
Function prototype	void timer_disable(uint32_t timer_periph);
Function descriptions	disable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2,5,15,16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */
timer_disable(TIMER0);
```

timer_auto_reload_shadow_enable

The description of timer_auto_reload_shadow_enable is shown as below:

Table 3-621. Function timer_auto_reload_shadow_enable

Function name	timer_auto_reload_shadow_enable
Function prototype	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
Function descriptions	enable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

timer_auto_reload_shadow_disable

The description of timer_auto_reload_shadow_disable is shown as below:

Table 3-622. Function timer_auto_reload_shadow_disable

Function name	timer_auto_reload_shadow_disable
Function prototype	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
Function descriptions	disable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

timer_update_event_enable

The description of timer_update_event_enable is shown as below:

Table 3-623. Function timer_update_event_enable

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 the update event */
timer_update_event_enable (TIMER0);
```

timer_update_event_disable

The description of timer_update_event_disable is shown as below:

Table 3-624. Function timer_update_event_disable

Function name	timer_update_event_disable
Function prototype	void timer_update_event_disable(uint32_t timer_periph);
Function descriptions	disable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable (TIMER0);
```

timer_counter_alignment

The description of timer_counter_alignment is shown as below:

Table 3-625. Function timer_counter_alignment

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	set TIMER counter alignment mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 1, 2)</i>	TIMER peripheral selection
Input parameter{in}	
aligned	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_COUNTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

timer_counter_up_direction

The description of timer_counter_up_direction is shown as below:

Table 3-626. Function timer_counter_up_direction

Function name	timer_counter_up_direction
Function prototype	void timer_counter_up_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter up direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter up direction */
timer_counter_up_direction(TIMER0);
```

timer_counter_down_direction

The description of timer_counter_down_direction is shown as below:

Table 3-627. timer_counter_down_direction

Function name	timer_counter_down_direction
Function prototype	void timer_counter_down_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter down direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter down direction */
timer_counter_down_direction(TIMER0);
```

timer_prescaler_config

The description of timer_prescaler_config is shown as below:

Table 3-628. Function timer_prescaler_config

Function name	timer_prescaler_config
Function prototype	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
Function descriptions	configure TIMER prescaler
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>	TIMER peripheral selection
Input parameter{in}	
prescaler	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

timer_repetition_value_config

The description of timer_repetition_value_config is shown as below:

Table 3-629. Function timer_repetition_value_config

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
repetition	the counter repetition value (0~255)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config(TIMER0, 98);
```

timer_autoreload_value_config

The description of timer_autoreload_value_config is shown as below:

Table 3-630. Function timer_autoreload_value_config

Function name	timer_autoreload_value_config
Function prototype	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 1, 2, 5, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
autoreload	the counter auto-reload value 0~65535(TIMERx(x=0, 5, 15, 16)), 0~4294967295(TIMERx(x=1, 2))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config(TIMER0, 3000);
```

timer_counter_value_config

The description of timer_counter_value_config is shown as below:

Table 3-631. Function timer_counter_value_config

Function name	timer_counter_value_config
Function prototype	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
Function descriptions	configure TIMER counter register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2,5,15,16)	TIMER peripheral selection
Input parameter{in}	
counter	the counter value0~65535(TIMERx(x=0,5,15,16)),0~4294967295(TIMERx(x=1,2))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0);
```

timer_counter_read

The description of timer_counter_read is shown as below:

Table 3-632. Function timer_counter_read

Function name	timer_counter_read
Function prototype	uint32_t timer_counter_read(uint32_t timer_periph);
Function descriptions	read TIMER counter value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2,5,15,16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint32_t	counter value(0~0xFFFFFFFF)

Example:

```
/* read TIMER0 counter value */
```

```
uint32_t i = 0;
```

```
i = timer_counter_read(TIMER0);
```

timer_prescaler_read

The description of timer_prescaler_read is shown as below:

Table 3-633. Function timer_prescaler_read

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER0);
```

timer_single_pulse_mode_config

The description of timer_single_pulse_mode_config is shown as below:

Table 3-634. Function timer_single_pulse_mode_config

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2,5,15,16)</i>)	TIMER peripheral selection
Input parameter{in}	

spmode	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

timer_update_source_config

The description of timer_update_source_config is shown as below:

Table 3-635. Function timer_update_source_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0,1,2,5,15,16)	TIMER peripheral selection
Input parameter{in}	
update	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: – The UPG bit is set – The counter generates an overflow or underflow event – The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMERO, TIMER_UPDATE_SRC_REGULAR);
```

timer_dma_enable

The description of timer_dma_enable is shown as below:

Table 3-636. Function timer_dma_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0,1,2,5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0,1,2)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0,1,2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0,1,2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMERO update DMA */
```

```
timer_dma_enable(TIMERO, TIMER_DMA_UPD);
```

timer_dma_disable

The description of timer_dma_disable is shown as below:

Table 3-637. Function timer_dma_disable

Function name	timer_dma_disable
Function prototype	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
Function descriptions	disable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, <i>TIMERx</i> (x=0,1,2,5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, <i>TIMERx</i> (x=0,1,2,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, <i>TIMERx</i> (x=0,1,2)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, <i>TIMERx</i> (x=0,1,2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, <i>TIMERx</i> (x=0,1,2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, <i>TIMERx</i> (x=0)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, <i>TIMERx</i> (x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

timer_channel_dma_request_source_select

The description of timer_channel_dma_request_source_select is shown as below:

Table 3-638. Function timer_channel_dma_request_source_select

Function name	timer_channel_dma_request_source_select
Function prototype	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
Function descriptions	channel DMA request source selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (x=0,1,2,15,16)	TIMER peripheral selection
Input parameter{in}	
dma_request	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel y is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel y is sent when update event occurs
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

```
timer_channel_dma_request_source_select(TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

timer_dma_transfer_config

The description of timer_dma_transfer_config is shown as below:

Table 3-639. Function timer_dma_transfer_config

Function name	timer_dma_transfer_config
Function prototype	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
Function descriptions	configure the TIMER DMA transfer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma_baseaddr	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0,1,2,15,16)

<i>TIMER_DMACFG_DMA</i> <i>TA_PSC</i>	DMA transfer address is <i>TIMER_PSC</i> , <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CAR</i>	MA transfer address is <i>TIMER_CAR</i> , <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CREP</i>	DMA transfer address is <i>TIMER_CREP</i> , <i>TIMERx</i> (<i>x</i> =0,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> (<i>x</i> =0,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMATB</i>	DMA transfer address is <i>TIMER_DMATB</i> , <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
Input parameter{in}	
<i>dma_lenth</i>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	<i>x</i> =1..18, DMA transfer <i>x</i> time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

timer_event_software_generate

The description of `timer_event_software_generate` is shown as below:

Table 3-640. Function `timer_event_software_generate`

Function name	<code>timer_event_software_generate</code>
Function prototype	<code>void timer_event_software_generate(uint32_t timer_periph, uint16_t event);</code>
Function descriptions	software generate events

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
event	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event,TIMERx(x=0,1,2,5,15,16)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation,TIMERx(x=0,1,2,15,16)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation,TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation,TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation,TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_COMG</i>	channel commutation event generation,TIMERx(x=0,15,16)
<i>TIMER_EVENT_SRC_TRIGG</i>	trigger event generation,TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_BREAKG</i>	break event generation,TIMERx(x=0,15,16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

timer_break_struct_para_init

The description of timer_break_struct_para_init is shown as below:

Table 3-641. Function timer_break_struct_para_init

Function name	timer_break_struct_para_init
Function prototype	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
Function descriptions	initialize the parameters of TIMER break parameter struct with the default values
Precondition	-

The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-613. Structure timer break parameter struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

timer_break_config

The description of timer_break_config is shown as below:

Table 3-642. Function timer_break_config

Function name	timer_break_config
Function prototype	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
Function descriptions	configure TIMER break function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0, 15, 16)	TIMER peripheral selection
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-613. Structure timer break parameter struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```
timer_breakpara.idleoffstate = TIMER_IOS_STATE_DISABLE ;
```

```

timer_breakpara.deadtime      = 255;

timer_breakpara.breakpolarity = TIMER_BREAK_POLARITY_LOW;

timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode    = TIMER_CCHP_PROT_0;

timer_breakpara.breakstate     = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

timer_break_enable

The description of timer_break_enable is shown as below:

Table 3-643. Function timer_break_enable

Function name	timer_break_enable
Function prototype	void timer_break_enable(uint32_t timer_periph);
Function descriptions	enable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable TIMER0 break function*/

timer_break_enable (TIMER0);

```

timer_break_disable

The description of timer_break_disable is shown as below:

Table 3-644. Function timer_break_disable

Function name	timer_break_disable
Function prototype	void timer_break_disable(uint32_t timer_periph);
Function descriptions	disable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMERO break function*/
```

```
timer_break_disable(TIMERO);
```

timer_automatic_output_enable

The description of timer_automatic_output_enable is shown as below:

Table 3-645. Function timer_automatic_output_enable

Function name	timer_automatic_output_enable
Function prototype	void timer_automatic_output_enable(uint32_t timer_periph);
Function descriptions	enable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMERO output automatic function */
```

```
timer_automatic_output_enable(TIMERO);
```

timer_automatic_output_disable

The description of timer_automatic_output_disable is shown as below:

Table 3-646. Function timer_automatic_output_disable

Function name	timer_automatic_output_disable
Function prototype	void timer_automatic_output_disable (uint32_t timer_periph);
Function descriptions	disable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in

	TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

timer_primary_output_config

The description of timer_primary_output_config is shown as below:

Table 3-647. Function timer_primary_output_config

Function name	timer_primary_output_config
Function prototype	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	configure TIMER primary output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config(TIMER0, ENABLE);
```

timer_channel_control_shadow_config

The description of timer_channel_control_shadow_config is shown as below:

Table 3-648. Function timer_channel_control_shadow_config

Function name	timer_channel_control_shadow_config
Function prototype	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	channel commutation control shadow register enable
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

timer_channel_control_shadow_update_config

The description of timer_channel_control_shadow_update_config is shown as below:

Table 3-649. Function timer_channel_control_shadow_update_config

Function name	timer_channel_control_shadow_update_config
Function prototype	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint32_t ccuctl);
Function descriptions	configure commutation control shadow register update control
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
ccuctl	channel control shadow register update control
<i>TIMER_UPDATECTL_C</i>	the shadow registers update by when CMTG bit is set

<i>CU</i>	
<i>TIMER_UPDATECTL_C</i> <i>CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

timer_channel_output_struct_para_init

The description of timer_channel_output_struct_para_init is shown as below:

Table 3-650. Function timer_channel_output_struct_para_init

Function name	timer_channel_output_struct_para_init
Function prototype	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
Function descriptions	initialize the parameters of TIMER channel output parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-614. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(&timer_ocinitpara);
```

timer_channel_output_config

The description of timer_channel_output_config is shown as below:

Table 3-651. Function timer_channel_output_config

Function name	timer_channel_output_config
----------------------	-----------------------------

Function prototype	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-614. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);

```

timer_channel_output_mode_config

The description of timer_channel_output_mode_config is shown as below:

Table 3-652. Function timer_channel_output_mode_config

Function name	timer_channel_output_mode_config
Function prototype	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t

	channel, uint16_t ocmode);
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocmode	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

timer_channel_output_pulse_value_config

The description of timer_channel_output_pulse_value_config is shown as below:

Table 3-653. Function timer_channel_output_pulse_value_config

Function name	timer_channel_output_pulse_value_config
Function prototype	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
Function descriptions	configure TIMER channel output pulse value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
pulse	channel output pulse value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

timer_channel_output_shadow_config

The description of timer_channel_output_shadow_config is shown as below:

Table 3-654. Function timer_channel_output_shadow_config

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured

<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocshadow	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

timer_channel_output_fast_config

The description of timer_channel_output_fast_config is shown as below:

Table 3-655. Function timer_channel_output_fast_config

Function name	timer_channel_output_fast_config
Function prototype	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
Function descriptions	configure TIMER channel output fast function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocfast	channel output fast function
<i>TIMER_OC_FAST_ENA</i>	channel output fast function enable

<i>BLE</i>	
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

timer_channel_output_clear_config

The description of timer_channel_output_clear_config is shown as below:

Table 3-656. Function timer_channel_output_clear_config

Function name	timer_channel_output_clear_config
Function prototype	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
Function descriptions	configure TIMER channel output clear function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
occlear	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

timer_channel_output_polarity_config

The description of timer_channel_output_polarity_config is shown as below:

Table 3-657. Function timer_channel_output_polarity_config

Function name	timer_channel_output_polarity_config
Function prototype	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
Function descriptions	configure TIMER channel output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocpolarity	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

timer_channel_complementary_output_polarity_config

The description of timer_channel_complementary_output_polarity_config is shown as below:

Table 3-658. Function timer_channel_complementary_output_polarity_config

Function name	timer_channel_complementary_output_polarity_config
Function prototype	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
Function descriptions	configure TIMER channel complementary output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0,15,16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0))
Input parameter{in}	
ocnpolarity	channel complementary output polarity
TIMER_OCN_POLARITY_HIGH	channel complementary output polarity is high
TIMER_OCN_POLARITY_LOW	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,
TIMER_OCN_POLARITY_HIGH);
```

timer_channel_output_state_config

The description of timer_channel_output_state_config is shown as below:

Table 3-659. Function timer_channel_output_state_config

Function name	timer_channel_output_state_config
Function prototype	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
Function descriptions	configure TIMER channel enable state
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
state	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

timer_channel_complementary_output_state_config

The description of timer_channel_complementary_output_state_config is shown as below:

Table 3-660. Function timer_channel_complementary_output_state_config

Function name	timer_channel_complementary_output_state_config
Function prototype	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
Function descriptions	configure TIMER channel complementary output enable state
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx(x=0,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx(x=0))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx(x=0))
Input parameter{in}	
ocnstate	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable

<i>TIMER_CCXN_DISABL</i> <i>E</i>	channel complementary disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

timer_channel_input_struct_para_init

The description of timer_channel_input_struct_para_init is shown as below:

Table 3-661. Function timer_channel_input_struct_para_init

Function name	timer_channel_input_struct_para_init
Function prototype	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
Function descriptions	initialize the parameters of TIMER channel input parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-615. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
timer_ic_parameter_struct timer_icinitpara;
timer_channel_input_struct_para_init(&timer_icinitpara);
```

timer_input_capture_config

The description of timer_input_capture_config is shown as below:

Table 3-662. Function timer_input_capture_config

Function name	timer_input_capture_config
----------------------	----------------------------

Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-615. Structure timer_ic_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

timer_channel_input_capture_prescaler_config

The description of timer_channel_input_capture_prescaler_config is shown as below:

Table 3-663. Function timer_channel_input_capture_prescaler_config

Function name	timer_channel_input_capture_prescaler_config
Function prototype	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
Function descriptions	configure TIMER channel input capture prescaler value
Precondition	-

The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
prescaler	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,  
TIMER_IC_PSC_DIV2);
```

timer_channel_capture_value_register_read

The description of timer_channel_capture_value_register_read is shown as below:

Table 3-664. Function timer_channel_capture_value_register_read

Function name	timer_channel_capture_value_register_read
Function prototype	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
Function descriptions	read TIMER channel capture compare register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))

<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Output parameter{out}	
-	-
Return value	
uint32_t	channel capture compare register value (0~0xFFFFFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

timer_input_pwm_capture_config

The description of timer_input_pwm_capture_config is shown as below:

Table 3-665. Function timer_input_pwm_capture_config

Function name	timer_input_pwm_capture_config
Function prototype	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
Function descriptions	configure TIMER input pwm capture function
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
Input parameter{in}	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to Table 3-615. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 input pwm capture parameter */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

timer_hall_mode_config

The description of timer_hall_mode_config is shown as below:

Table 3-666. Function timer_hall_mode_config

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
Function descriptions	configure TIMER hall sensor mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2)	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFACE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFACE_DISABLE	TIMER hall sensor mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 hall sensor mode */

timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);

```

timer_channel_input_remap_config

The description of timer_channel_input_remap_config is shown as below:

Table 3-667. Function timer_channel_input_remap_config

Function name	timer_channel_input_remap_config
Function prototype	void timer_channel_input_remap_config(uint32_t timer_periph, uint32_t remap);

Function descriptions	configure TIMER channel input remap
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=2)</i>	TIMER peripheral selection
Input parameter{in}	
remap	TIMER channel input remap selection
<i>TIMER_IRMP_GPIO</i>	channel 3 input is connected to GPIO
<i>TIMER_IRMP_IRC32K</i>	channel 3 input is connected to IRC32K
<i>TIMER_IRMP_LXTAL</i>	channel 3 input is connected to LXTAL
<i>TIMER_IRMP_CKOUT</i>	channel 3 input is connected to CKOUT
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER channel input remap */
```

```
timer_channel_input_remap_config(TIMER2, TIMER_IRMP_GPIO);
```

timer_input_trigger_source_select

The description of timer_input_trigger_source_select is shown as below:

Table 3-668. Function timer_input_trigger_source_select

Function name	timer_input_trigger_source_select
Function prototype	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	select TIMER input trigger source
Precondition	SMC[2:0] = 000
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 1
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 2

<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 3
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

timer_master_output_trigger_source_select

The description of timer_master_output_trigger_source_select is shown as below:

Table 3-669. Function timer_master_output_trigger_source_select

Function name	timer_master_output_trigger_source_select
Function prototype	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
Function descriptions	select TIMER master mode output trigger source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
outrigger	master mode control
<i>TIMER_TRI_OUT_SRC</i> <i>_RESET</i>	Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC</i> <i>_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause

	mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CC0</i>	Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

timer_slave_mode_select

The description of timer_slave_mode_select is shown as below:

Table 3-670. Function timer_slave_mode_select

Function name	timer_slave_mode_select
Function prototype	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1

<i>ER_MODE1</i>	
<i>TIMER_QUAD_DECODE</i> <i>ER_MODE2</i>	quadrature decoder mode 2
<i>TIMER_SLAVE_MODE</i> <i>RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE</i> <i>PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE</i> <i>EVENT</i>	event mode
<i>TIMER_SLAVE_MODE</i> <i>EXTERNAL0</i>	external clock mode 0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

timer_master_slave_mode_config

The description of timer_master_slave_mode_config is shown as below:

Table 3-671. Function timer_master_slave_mode_config

Function name	timer_master_slave_mode_config
Function prototype	void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);
Function descriptions	configure TIMER master slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
masterslave	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

timer_external_trigger_config

The description of timer_external_trigger_config is shown as below:

Table 3-672. Function timer_external_trigger_config

Function name	timer_external_trigger_config
Function prototype	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER external trigger input
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

timer_quadrature_decoder_mode_config

The description of timer_quadrature_decoder_mode_config is shown as below:

Table 3-673. Function timer_quadrature_decoder_mode_config

Function name	timer_quadrature_decoder_mode_config
Function prototype	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
Function descriptions	configure TIMER quadrature decoder mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
decomode	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
ic0polarity	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Input parameter{in}	
ic1polarity	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TIMER0 quadrature decoder mode */

timer_quadrature_decoder_mode_config(TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

timer_internal_clock_config

The description of timer_internal_clock_config is shown as below:

Table 3-674. Function timer_internal_clock_config

Function name	timer_internal_clock_config
Function prototype	void timer_internal_clock_config(uint32_t timer_periph);
Function descriptions	configure TIMER internal clock mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 internal clock mode */

timer_internal_clock_config (TIMER0);
```

timer_internal_trigger_as_external_clock_config

The description of timer_internal_trigger_as_external_clock_config is shown as below:

Table 3-675. Function timer_internal_trigger_as_external_clock_config

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	configure TIMER the internal trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection

Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

timer_external_trigger_as_external_clock_config

The description of timer_external_trigger_as_external_clock_config is shown as below:

Table 3-676. Function timer_external_trigger_as_external_clock_config

Function name	timer_external_trigger_as_external_clock_config
Function prototype	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
extrigger	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1)
Input parameter{in}	
expolarity	external trigger polarity

<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

timer_external_clock_mode0_config

The description of timer_external_clock_mode0_config is shown as below:

Table 3-677. Function timer_external_clock_mode0_config

Function name	timer_external_clock_mode0_config
Function prototype	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode0
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	

expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_config

The description of timer_external_clock_mode1_config is shown as below:

Table 3-678. Function timer_external_clock_mode1_config

Function name	timer_external_clock_mode1_config
Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t extpolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
extpolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active

Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_disable

The description of timer_external_clock_mode1_disable is shown as below:

Table 3-679. Function timer_external_clock_mode1_disable

Function name	timer_external_clock_mode1_disable
Function prototype	void timer_external_clock_mode1_disable(uint32_t timer_periph);
Function descriptions	disable TIMER the external clock mode1
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER0);
```

timer_write_chxval_register_config

The description of timer_write_chxval_register_config is shown as below:

Table 3-680. Function timer_write_chxval_register_config

Function name	timer_write_chxval_register_config
Function prototype	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
Function descriptions	configure TIMER write CHxVAL register selection

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0, 1, 2, 15, 16)	TIMER peripheral selection
Input parameter{in}	
ccsel	write CHxVAL register selection
<i>TIMER_CHVSEL_DISABLE</i>	no effect
<i>TIMER_CHVSEL_ENABLE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

timer_output_value_selection_config

The description of timer_output_value_selection_config is shown as below:

Table 3-681. Function timer_output_value_selection_config

Function name	timer_output_value_selection_config
Function prototype	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
Function descriptions	configure TIMER output value selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0, 15, 16)	TIMER peripheral selection
Input parameter{in}	
outsel	output value selection
<i>TIMER_OUTSEL_DISABLE</i>	no effect
<i>TIMER_OUTSEL_ENABLE</i>	if POEN and IOS is 0, the output disabled
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-682. Function timer_flag_get

Function name	timer_flag_get
Function prototype	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
Function descriptions	get TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0,1,2,5,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

timer_flag_clear

The description of timer_flag_clear is shown as below:

Table 3-683. Function timer_flag_clear

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0,1,2,5,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

timer_interrupt_enable

The description of timer_interrupt_enable is shown as below:

Table 3-684. Function timer_interrupt_enable

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0,1,2,5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0,1,2,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx (x=0,1,2)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,15,16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update interrupt */
```

```
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

timer_interrupt_disable

The description of timer_interrupt_disable is shown as below:

Table 3-685. Function timer_interrupt_disable

Function name	timer_interrupt_disable
Function prototype	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
Function descriptions	disable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0,1,2,5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0,1,2,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0,1,2)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0,1,2)

<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, <i>TIMERx</i> (<i>x</i> =0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_BRK</i>	break interrupt disable, <i>TIMERx</i> (<i>x</i> =0,15,16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

timer_interrupt_flag_get

The description of timer_interrupt_flag_get is shown as below:

Table 3-686. Function timer_interrupt_flag_get

Function name	timer_interrupt_flag_get
Function prototype	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
Function descriptions	get timer interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
int_flag	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2,5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CMT</i> <i>T</i>	channel commutation interrupt flag, <i>TIMERx</i> (<i>x</i> =0,15,16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> (<i>x</i> =0,15,16)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update interrupt flag */

FlagStatus Flag_interrupt = RESET;

Flag_interrupt = timer_interrupt_flag_get(TIMER0, TIMER_INT_FLAG_UP);
```

timer_interrupt_flag_clear

The description of timer_interrupt_flag_clear is shown as below:

Table 3-687. Function timer_interrupt_flag_clear

Function name	timer_interrupt_flag_clear
Function prototype	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
Function descriptions	clear TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
int_flag	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2,5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2,15,16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> (<i>x</i> =0,15,16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> (<i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> (<i>x</i> =0,15,16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */

timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.23. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using

continuous analog noise. The TRNG registers are listed in chapter [3.23.1](#). the TRNG firmware functions are introduced in chapter [3.23.2](#).

3.23.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

Table 3-688 TRNG Registers

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

3.23.2. Descriptions of Peripheral functions

TRNG firmware functions are listed in the table shown as below:

Table 3-689. TRNG firmware function

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_flag_get	get the TRNG status flags
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

Enum trng_flag_enum

Table 3-690. Enum trng_flag_enum

Member name	Function description
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status

Enum trng_int_flag_enum

Table 3-691. Enum trng_int_flag_enum

Member name	Function description
TRNG_INT_FLAG_CEIF	clock error interrupt flag
TRNG_INT_FLAG_SEIF	seed error interrupt flag

trng_deinit

The description of trng_deinit is shown as below:

Table 3-692. Function trng_deinit

Function name	trng_deinit
Function prototype	void trng_deinit(void)
Function descriptions	deinit the TRNG
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TRNG */
trng_deinit();
```

trng_enable

The description of trng_enable is shown as below:

Table 3-693. Function trng_enable

Function name	trng_enable
Function prototype	void trng_enable(void)
Function descriptions	enable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
trng_enable();
```

trng_disable

The description of trng_disable is shown as below:

Table 3-694 Function trng_disable

Function name	trng_disable
Function prototype	void trng_disable(void);
Function descriptions	disable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG */
trng_disable();
```

trng_get_true_random_data

The description of trng_get_true_random_data is shown as below:

Table 3-695 Function trng_get_true_random_data

Function name	trng_get_true_random_data
Function prototype	uint32_t trng_get_true_random_data(void)
Function descriptions	get the true random data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the generated random data

Example:

```
/* get true random data */
uint32_t data;
data = trng_get_true_random_data();
```


trng_interrupt_enable

The description of trng_interrupt_enable is shown as below:

Table 3-696 Function trng_interrupt_enable

Function name	trng_interrupt_enable
Function prototype	void trng_interrupt_enable(void);
Function descriptions	enable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

trng_interrupt_disable

The description of trng_interrupt_disable is shown as below:

Table 3-697 Function trng_interrupt_disable

Function name	trng_interrupt_disable
Function prototype	void trng_interrupt_disable(void);
Function descriptions	disable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

trng_flag_get

The description of trng_flag_get is shown as below:

Table 3-698 Function trng_flag_get

Function name	trng_flag_get
Function prototype	FlagStatus trng_flag_get(trng_flag_enum flag);
Function descriptions	get the trng status flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	trng_flag_enum, please refer to Table 3-690. Enum trng_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error current flag status */
FlagStatus flag_status = RESET;
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

trng_interrupt_flag_get

The description of trng_interrupt_flag_get is shown as below:

Table 3-699 Function trng_interrupt_flag_get

Function name	trng_interrupt_flag_get
Function prototype	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);
Function descriptions	get the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	trng_int_flag_enum, please refer to Table 3-691. Enum trng_int_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag */
FlagStatus interrupt_flag = RESET;
```

```
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

trng_interrupt_flag_clear

The description of trng_interrupt_flag_clear is shown as below:

Table 3-700 Function trng_interrupt_flag_clear

Function name	trng_interrupt_flag_clear
Function prototype	void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);
Function descriptions	clear the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	trng_flag_enum, please refer to Table 3-691. Enum trng_int_flag_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TRNG clock error interrupt flag */

trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

3.24. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.24.1](#), the USART firmware functions are introduced in chapter [3.24.2](#).

3.24.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

Table 3-701. USART Registers

Registers	Descriptions
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_BAUD	Baud rate register
USART_GP	Guard time and prescaler register
USART_RT	Receiver timeout register
USART_CMD	Command register
USART_STAT	Status register

Registers	Descriptions
USART_INTIC	Status clear register
USART_RDATA	Receive data register
USART_TDATA	Transmit data register
USART_CHC	Coherence control register
USART_RFCS	Receive FIFO control and status register

3.24.2. Descriptions of Peripheral functions

USART firmware functions are listed in the table shown as below:

Table 3-702. USART firmware function

Function name	Function description
usart_deinit	reset USART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity function
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inverted
usart_overrun_enable	enable the USART overrun function
usart_overrun_disable	disable the USART overrun function
usart_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_command_enable	enable USART command
usart_address_config	configure address of the USART
usart_address_detection_mode_config	configure address detection mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode

Function name	Function description
usart_lin_break_detection_length_config	LIN break detection length
usart_halfduplex_enable	enable half-duplex mode
usart_halfduplex_disable	disable half-duplex mode
usart_clock_enable	enable clock
usart_clock_disable	disable clock
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_mode_early_nack_enable	enable early NACK in smartcard mode
usart_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power or SmartCard mode
usart_irda_lowpower_config	configure IrDA low-power
usart_hardware_flow_rts_config	configure hardware flow control RTS
usart_hardware_flow_cts_config	configure hardware flow control CTS
usart_hardware_flow_coherence_config	configure hardware flow control coherence mode
usart_rs485_driver_enable	enable RS485 driver
usart_rs485_driver_disable	disable RS485 driver
usart_driver_asserttime_config	configure driver enable assertion time
usart_driver_deasserttime_config	configure driver enable de-assertion time
usart_depolarity_config	configure driver enable polarity mode
usart_dma_receive_config	configure USART DMA reception
usart_dma_transmit_config	configure USART DMA transmission
usart_reception_error_dma_disable	disable DMA on reception error
usart_reception_error_dma_enable	enable DMA on reception error
usart_wakeup_enable	enable USART to wakeup the MCU from deep-sleep mode
usart_wakeup_disable	disable USART to wakeup the MCU from deep-sleep mode
usart_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode
usart_receive_fifo_enable	enable receive FIFO
usart_receive_fifo_disable	disable receive FIFO

Function name	Function description
usart_receive_fifo_counter_number	read receive FIFO counter number
usart_flag_get	get flag in STAT/RFCR register
usart_flag_clear	clear USART status
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt and flag status
usart_interrupt_flag_clear	clear USART interrupt flag

Enum usart_flag_enum

Table 3-703. Enum usart_flag_enum

Member name	Function description
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_WU	wakeup from Deep-sleep mode flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_SB	send break flag
USART_FLAG_AM	ADDR match flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFFINT	receive FIFO full interrupt flag
USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFE	receive FIFO empty flag

Enum usart_interrupt_flag_enum

Table 3-704. Enum usart_interrupt_flag_enum

Member name	Function description
USART_INT_FLAG_EB	end of block interrupt and flag

Member name	Function description
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_AM	address match interrupt and flag
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_RFF	receive FIFO full interrupt and flag

Enum usart_interrupt_enum

Table 3-705. Enum usart_interrupt_enum

Member name	Function description
USART_INT_EB	end of block interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_AM	address match interrupt
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_RFF	receive FIFO full interrupt

Enum usart_invert_enum

Table 3-706. Enum usart_invert_enum

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion

Member name	Function description
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion
USART_SWAP_ENABLE	swap TX/RX pins
USART_SWAP_DISABLE	not swap TX/RX pins

usart_deinit

The description of usart_deinit is shown as below:

Table 3-707. Function usart_deinit

Function name	usart_deinit
Function prototype	void usart_deinit(uint32_t usart_periph);
Function descriptions	reset USART
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset USART0 */
usart_deinit (USART0);
```

usart_baudrate_set

The description of usart_baudrate_set is shown as below:

Table 3-708. Function usart_baudrate_set

Function name	usart_baudrate_set
Function prototype	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
Function descriptions	configure USART baud rate value
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	

baudval	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

usart_parity_config

The description of usart_parity_config is shown as below:

Table 3-709. Function usart_parity_config

Function name	usart_parity_config
Function prototype	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
Function descriptions	configure USART parity function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
paritycfg	USART parity configure
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	even parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

usart_word_length_set

The description of usart_word_length_set is shown as below:

Table 3-710. Function usart_word_length_set

Function name	usart_word_length_set
----------------------	-----------------------

Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
wlen	USART word length configure
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

usart_stop_bit_set

The description of usart_stop_bit_set is shown as below:

Table 3-711. Function usart_stop_bit_set

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
Function descriptions	configure USART stop bit length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
stblen	USART stop bit configure
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 stop bit length */

usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

usart_enable

The description of usart_enable is shown as below:

Table 3-712. Function usart_enable

Function name	usart_enable
Function prototype	void usart_enable(uint32_t usart_periph);
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 */

usart_enable(USART0);
```

usart_disable

The description of usart_disable is shown as below:

Table 3-713. Function usart_disable

Function name	usart_disable
Function prototype	void usart_disable(uint32_t usart_periph);
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

usart_transmit_config

The description of usart_transmit_config is shown as below:

Table 3-714. Function usart_transmit_config

Function name	usart_transmit_config
Function prototype	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
Function descriptions	configure USART transmitter
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
txconfig	enable or disable USART transmitter
USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	disable USART transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

usart_receive_config

The description of usart_receive_config is shown as below:

Table 3-715. Function usart_receive_config

Function name	usart_receive_config
Function prototype	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
Function descriptions	configure USART receiver
Precondition	-
The called functions	-
Input parameter{in}	

usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
rxconfig	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	disable USART reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

usart_data_first_config

The description of usart_data_first_config is shown as below:

Table 3-716. Function usart_data_first_config

Function name	usart_data_first_config
Function prototype	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
msbf	LSB/MSB
<i>USART_MSBF_LSB</i>	LSB first
<i>USART_MSBF_MSB</i>	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 data is transmitted/received with the LSB first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

usart_invert_config

The description of usart_invert_config is shown as below:

Table 3-717. Function usart_invert_config

Function name	usart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
Function descriptions	USART inverted configure
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
invertpara	refer to Table 3-706. Enum usart_invert_enum
USART_DINV_ENAB LE	data bit level inversion
USART_DINV_DISAB LE	data bit level not inversion
USART_TXPIN_ENAB LE	TX pin level inversion
USART_TXPIN_DISAB LE	TX pin level not inversion
USART_RXPIN_ENAB LE	RX pin level inversion
USART_RXPIN_DISAB LE	RX pin level not inversion
USART_SWAP_ENAB LE	swap TX/RX pins
USART_SWAP_DISAB LE	not swap TX/RX pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 data bit level inversion */
```

```
usart_invert_config (USART0, USART_DINV_ENABLE);
```

usart_oversize_enable

The description of usart_oversize_enable is shown as below:

Table 3-718. Function usart_oversize_enable

Function name	usart_oversize_enable
Function prototype	void usart_oversize_enable(uint32_t usart_periph);
Function descriptions	enable the USART oversize function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 oversize function */
```

```
usart_oversize_enable(USART0);
```

usart_oversize_disable

The description of usart_oversize_disable is shown as below:

Table 3-719. Function usart_oversize_disable

Function name	usart_oversize_disable
Function prototype	void usart_oversize_disable(uint32_t usart_periph);
Function descriptions	disable the USART oversize function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the USART0 oversize function */
```

```
usart_oversize_disable(USART0);
```

usart_oversample_config

The description of usart_oversample_config is shown as below:

Table 3-720. Function usart_oversample_config

Function name	usart_oversample_config
Function prototype	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
Function descriptions	configure the USART oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
oversamp	oversample value
USART_OVSMOD_8	8 bits
USART_OVSMOD_16	16 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 oversample mode */
```

```
usart_oversample_config(USART0, USART_OVSMOD_8);
```

usart_sample_bit_config

The description of usart_sample_bit_config is shown as below:

Table 3-721. Function usart_sample_bit_config

Function name	usart_sample_bit_config
Function prototype	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
Function descriptions	configure the sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
osb	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure USART0 sample bit */
```

```
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

usart_receiver_timeout_enable

The description of usart_receiver_timeout_enable is shown as below:

Table 3-722. Function usart_receiver_timeout_enable

Function name	usart_receiver_timeout_enable
Function prototype	void usart_receiver_timeout_enable(uint32_t usart_periph);
Function descriptions	enable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receiver timeout of USART */
```

```
usart_receiver_timeout_enable(USART0);
```

usart_receiver_timeout_disable

The description of usart_receiver_timeout_disable is shown as below:

Table 3-723. Function usart_receiver_timeout_disable

Function name	usart_receiver_timeout_disable
Function prototype	void usart_receiver_timeout_disable(uint32_t usart_periph);
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receiver timeout of USART */
```

```
usart_receiver_timeout_disable(USART0);
```

usart_receiver_timeout_threshold_config

The description of usart_receiver_timeout_threshold_config is shown as below:

Table 3-724. Function usart_receiver_timeout_threshold_config

Function name	usart_receiver_timeout_threshold_config
Function prototype	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t timeout);
Function descriptions	configure receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0	USART0
Input parameter{in}	
rtimeout	receiver timeout threshold (0x00000000 - 0x00FFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

usart_data_transmit

The description of usart_data_transmit is shown as below:

Table 3-725. Function usart_data_transmit

Function name	usart_data_transmit
Function prototype	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
Function descriptions	USART transmit data function
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
data	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

usart_data_receive

The description of usart_data_receive is shown as below:

Table 3-726. Function usart_data_receive

Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(uint32_t usart_periph);
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
uint16_t	data of received

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

usart_command_enable

The description of usart_command_enable is shown as below:

Table 3-727. Function `usart_command_enable`

Function name	<code>usart_command_enable</code>
Function prototype	<code>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
Function descriptions	enable USART command
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
cmdtype	command type
<i>USART_CMD_SBKCMD</i> <i>D</i>	send break command
<i>USART_CMD_MMCMMD</i>	mute mode command
<i>USART_CMD_RXFCM</i> <i>D</i>	receive data flush command
<i>USART_CMD_TXFCM</i> <i>D</i>	transmit data flush request
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

`usart_address_config`

The description of `usart_address_config` is shown as below:

Table 3-728. Function `usart_address_config`

Function name	<code>usart_address_config</code>
Function prototype	<code>void usart_address_config(uint32_t usart_periph, uint8_t addr);</code>
Function descriptions	address of the USART terminal
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
addr	address of USART terminal(0x00-0xFF)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure address of USART0 */
```

```
usart_address_config(USART0, 0x00);
```

usart_address_detection_mode_config

The description of usart_address_detection_mode_config is shown as below:

Table 3-729. Function usart_address_detection_mode_config

Function name	usart_address_detection_mode_config
Function prototype	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
Function descriptions	configure address detection mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
addmod	address detection mode
<i>USART_ADDDM_4BIT</i>	4 bits
<i>USART_ADDDM_FULLBIT</i>	full bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address detection mode */
```

```
usart_address_config(USART0, USART_ADDDM_4BIT);
```

usart_mute_mode_enable

The description of usart_mute_mode_enable is shown as below:

Table 3-730. Function usart_mute_mode_enable

Function name	usart_mute_mode_enable
Function prototype	void usart_mute_mode_enable(uint32_t usart_periph);

Function descriptions	enable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

usart_mute_mode_disable

The description of usart_mute_mode_disable is shown as below:

Table 3-731. Function usart_mute_mode_disable

Function name	usart_mute_mode_disable
Function prototype	void usart_mute_mode_disable(uint32_t usart_periph);
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

usart_mute_mode_wakeup_config

The description of usart_mute_mode_wakeup_config is shown as below:

Table 3-732. Function usart_mute_mode_wakeup_config

Function name	usart_mute_mode_wakeup_config
----------------------	-------------------------------

Function prototype	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
wmethod	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mark
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

usart_lin_mode_enable

The description of usart_lin_mode_enable is shown as below:

Table 3-733. Function usart_lin_mode_enable

Function name	usart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(uint32_t usart_periph);
Function descriptions	enable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

usart_lin_mode_disable

The description of usart_lin_mode_disable is shown as below:

Table 3-734. Function usart_lin_mode_disable

Function name	usart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

usart_lin_break_dection_length_config

The description of usart_lin_break_dection_length_config is shown as below:

Table 3-735. Function usart_lin_break_dection_length_config

Function name	usart_lin_break_dection_length_config
Function prototype	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
Function descriptions	configure LIN break detection length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Input parameter{in}	
lblen	LIN break detection length
<i>USART_LBLEN_10B</i>	10 bits break detection
<i>USART_LBLEN_11B</i>	11 bits break detection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */

usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

usart_halfduplex_enable

The description of usart_halfduplex_enable is shown as below:

Table 3-736. Function usart_halfduplex_enable

Function name	usart_halfduplex_enable
Function prototype	void usart_halfduplex_enable(uint32_t usart_periph);
Function descriptions	enable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode */

usart_halfduplex_enable(USART0);
```

usart_halfduplex_disable

The description of usart_halfduplex_disable is shown as below:

Table 3-737. Function usart_halfduplex_disable

Function name	usart_halfduplex_disable
Function prototype	void usart_halfduplex_disable(uint32_t usart_periph);
Function descriptions	disable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/

usart_halfduplex_disable(USART0);
```

usart_clock_enable

The description of usart_clock_enable is shown as below:

Table 3-738. Function usart_clock_enable

Function name	usart_clock_enable
Function prototype	void usart_clock_enable(uint32_t usart_periph);
Function descriptions	enable clock
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */

usart_clock_enable(USART0);
```

usart_clock_disable

The description of usart_clock_disable is shown as below:

Table 3-739. Function usart_clock_disable

Function name	usart_clock_disable
Function prototype	void usart_clock_disable(uint32_t usart_periph);
Function descriptions	disable clock
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_clock_disable(USART0);
```

usart_synchronous_clock_config

The description of usart_synchronous_clock_config is shown as below:

Table 3-740. Function usart_synchronous_clock_config

Function name	usart_synchronous_clock_config
Function prototype	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0	USART0
Input parameter{in}	
clen	CK length
USART_CLEN_NONE	clock pulse of the last data bit (MSB) is not output to the CK pin
USART_CLEN_EN	clock pulse of the last data bit (MSB) is output to the CK pin
Input parameter{in}	
cph	clock phase
USART_CPH_1CK	first clock transition is the first data capture edge
USART_CPH_2CK	second clock transition is the first data capture edge
Input parameter{in}	
cpl	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,    USART_CLEN_EN,    USART_CPH_2CK,
USART_CPL_HIGH);
```

usart_guard_time_config

The description of usart_guard_time_config is shown as below:

Table 3-741. Function `usart_guard_time_config`

Function name	<code>usart_guard_time_config</code>
Function prototype	<code>void usart_guard_time_config(uint32_t usart_periph, uint32_t gaut);</code>
Function descriptions	configure guard time value in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USART0/UARTx peripheral
<code>USART0</code>	USART0
Input parameter{in}	
<code>gaut</code>	guard time value (0x00 - 0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
usart_guard_time_config(USART0, 0x55);
```

`usart_smartcard_mode_enable`

The description of `usart_smartcard_mode_enable` is shown as below:

Table 3-742. Function `usart_smartcard_mode_enable`

Function name	<code>usart_smartcard_mode_enable</code>
Function prototype	<code>void usart_smartcard_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USART0/UARTx peripheral
<code>USART0</code>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);
```

usart_smartcard_mode_disable

The description of usart_smartcard_mode_disable is shown as below:

Table 3-743. Function usart_smartcard_mode_disable

Function name	usart_smartcard_mode_disable
Function prototype	void usart_smartcard_mode_disable(uint32_t usart_periph);
Function descriptions	disable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

usart_smartcard_mode_nack_enable

The description of usart_smartcard_mode_nack_enable is shown as below:

Table 3-744. Function usart_smartcard_mode_nack_enable

Function name	usart_smartcard_mode_nack_enable
Function prototype	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

usart_smartcard_mode_nack_disable

The description of usart_smartcard_mode_nack_disable is shown as below:

Table 3-745. Function usart_smartcard_mode_nack_disable

Function name	usart_smartcard_mode_nack_disable
Function prototype	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
Function descriptions	disable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

usart_smartcard_mode_early_nack_enable

The description of usart_smartcard_mode_early_nack_enable is shown as below:

Table 3-746. Function usart_smartcard_mode_early_nack_enable

Function name	usart_smartcard_mode_early_nack_enable
Function prototype	void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
Function descriptions	enable early NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);
```

usart_smartcard_mode_early_nack_disable

The description of usart_smartcard_mode_early_nack_disable is shown as below:

Table 3-747. Function usart_smartcard_mode_early_nack_disable

Function name	usart_smartcard_mode_early_nack_disable
Function prototype	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
Function descriptions	disable early NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 early NACK in smartcard mode */
```

```
usart_smartcard_mode_early_nack_disable(USART0);
```

usart_smartcard_autoretry_config

The description of usart_smartcard_autoretry_config is shown as below:

Table 3-748. Function usart_smartcard_autoretry_config

Function name	usart_smartcard_autoretry_config
Function prototype	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
Function descriptions	configure smartcard auto-retry number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Input parameter{in}	
scrtnum	smartcard auto-retry number (0x00 - 0x07)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config (USART0, 0x07);
```

usart_block_length_config

The description of usart_block_length_config is shown as below:

Table 3-749. Function usart_block_length_config

Function name	usart_block_length_config
Function prototype	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
Function descriptions	configure block length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Input parameter{in}	
bl	block length (0x00 - 0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

usart_irda_mode_enable

The description of usart_irda_mode_enable is shown as below:

Table 3-750. Function usart_irda_mode_enable

Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(uint32_t usart_periph);
Function descriptions	enable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

usart_irda_mode_disable

The description of usart_irda_mode_disable is shown as below:

Table 3-751. Function usart_irda_mode_disable

Function name	usart_irda_mode_disable
Function prototype	void usart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

usart_prescaler_config

The description of usart_prescaler_config is shown as below:

Table 3-752. Function usart_prescaler_config

Function name	usart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power or smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Input parameter{in}	

psc	0x00000000 - 0x000000FF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00000000);
```

usart_irda_lowpower_config

The description of usart_irda_lowpower_config is shown as below:

Table 3-753. Function usart_irda_lowpower_config

Function name	usart_irda_lowpower_config
Function prototype	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
Function descriptions	configure IrDA low-power
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0</i>	USART0
Input parameter{in}	
irlp	IrDA low-power or normal
<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMAL</i>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

usart_hardware_flow_rts_config

The description of usart_hardware_flow_rts_config is shown as below:

Table 3-754. Function usart_hardware_flow_rts_config

Function name	usart_hardware_flow_rts_config
----------------------	--------------------------------

Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
rtsconfig	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	disable RTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

usart_hardware_flow_cts_config

The description of usart_hardware_flow_cts_config is shown as below:

Table 3-755. Function usart_hardware_flow_cts_config

Function name	usart_hardware_flow_cts_config
Function prototype	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
Function descriptions	configure hardware flow control CTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
ctsconfig	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

usart_hardware_flow_coherence_config

The description of usart_hardware_flow_coherence_config is shown as below:

Table 3-756. Function usart_hardware_flow_coherence_config

Function name	usart_hardware_flow_coherence_config
Function prototype	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
Function descriptions	configure hardware flow control coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
hcm	hardware flow control coherence mode
USART_HCM_NONE	nRTS signal equals to the rbne status register
USART_HCM_EN	nRTS signal is set when the last data bit has been sampled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

usart_rs485_driver_enable

The description of usart_rs485_driver_enable is shown as below:

Table 3-757. Function usart_rs45_driver_enable

Function name	usart_rs485_driver_enable
Function prototype	void usart_rs485_driver_enable(uint32_t usart_periph);
Function descriptions	enable RS485 driver
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 RS485 driver */
```

```
usart_rs485_driver_enable(USART0);
```

usart_rs485_driver_disable

The description of usart_rs485_driver_disable is shown as below:

Table 3-758. Function usart_rs45_driver_disable

Function name	usart_rs485_driver_disable
Function prototype	void usart_rs485_driver_disable(uint32_t usart_periph);
Function descriptions	disable RS485 driver
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 RS485 driver */
```

```
usart_rs485_driver_disable(USART0);
```

usart_driver_assertime_config

The description of usart_driver_assertime_config is shown as below:

Table 3-759. Function usart_driver_assertime_config

Function name	usart_driver_assertime_config
Function prototype	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);

Function descriptions	configure driver enable assertion time
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
deatime	driver enable assertion time(0x00-0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x1F);
```

usart_driver_deassertime_config

The description of usart_driver_deassertime_config is shown as below:

Table 3-760. Function usart_driver_deassertime_config

Function name	usart_driver_deassertime_config
Function prototype	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
Function descriptions	configure driver enable de-assertion time
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
deatime	driver enable deassertime (0x00-0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set USART0 driver deassertime */
```

```
usart_driver_deassertime_config(USART0, 0x1F);
```

usart_depolarity_config

The description of usart_depolarity_config is shown as below:

Table 3-761. Function usart_depolarity_config

Function name	usart_depolarity_config
Function prototype	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
Function descriptions	configure driver enable polarity mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
dep	DE signal
USART_DEP_HIGH	DE signal is active high
USART_DEP_LOW	DE signal is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

usart_dma_receive_config

The description of usart_dma_receive_config is shown as below:

Table 3-762. Function usart_dma_receive_config

Function name	usart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
Function descriptions	configure USART DMA reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
dmacmd	USART DMA mode
USART_RECEIVE_DMA_ENABLE	enable USART DMA for reception
USART_RECEIVE_DMA_DISABLE	disable USART DMA for reception

<i>A_DISABLE</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART DMA reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

usart_dma_transmit_config

The description of usart_dma_transmit_config is shown as below:

Table 3-763. Function usart_dma_transmit_config

Function name	usart_dma_transmit_config
Function prototype	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
Function descriptions	configure USART DMA transmission
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
dmacmd	USART DMA mode
<i>USART_TRANSMIT_DMA_ENABLE</i>	enable USART DMA for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	disable USART DMA for transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART DMA transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

usart_reception_error_dma_disable

The description of usart_reception_error_dma_disable is shown as below:

Table 3-764. Function `usart_reception_error_dma_disable`

Function name	<code>usart_reception_error_dma_disable</code>
Function prototype	<code>void usart_reception_error_dma_disable(uint32_t usart_periph);</code>
Function descriptions	disable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA on reception error */
usart_reception_error_dma_disable (USART0);
```

`usart_reception_error_dma_enable`

The description of `usart_reception_error_dma_enable` is shown as below:

Table 3-765. Function `usart_reception_error_dma_enable`

Function name	<code>usart_reception_error_dma_enable</code>
Function prototype	<code>void usart_reception_error_dma_enable(uint32_t usart_periph);</code>
Function descriptions	enable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA on reception error */
usart_reception_error_dma_enable (USART0);
```

`usart_wakeup_enable`

The description of `usart_wakeup_enable` is shown as below:

Table 3-766. Function usart_wakeup_enable

Function name	usart_wakeup_enable
Function prototype	void usart_wakeup_enable(uint32_t usart_periph);
Function descriptions	enable USART to wakeup the MCU from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```

usart_wakeup_disable

The description of usart_wakeup_disable is shown as below:

Table 3-767. Function usart_wakeup_disable

Function name	usart_wakeup_disable
Function prototype	void usart_wakeup_disable(uint32_t usart_periph);
Function descriptions	disable USART to wakeup the MCU from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0	USART0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up disable */
usart_wakeup_disable(USART0);
```

usart_wakeup_mode_config

The description of usart_wakeup_mode_config is shown as below:

Table 3-768. Function usart_wakeup_mode_config

Function name	usart_wakeup_mode_config
Function prototype	void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
Function descriptions	configure the USART wakeup mode from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0	USART0
Input parameter{in}	
wum	wakeup mode
USART_WUM_ADDR	WUF active on address match
USART_WUM_START B	WUF active on start bit
USART_WUM_RBNE	WUF active on RBNE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART wake up mode */
```

```
usart_wakeup_mode_config(USART, USART_WUM_ADDR);
```

usart_receive_fifo_enable

The description of usart_receive_fifo_enable is shown as below:

Table 3-769. Function usart_receive_fifo_enable

Function name	usart_receive_fifo_enable
Function prototype	void usart_receive_fifo_enable(uint32_t usart_periph);
Function descriptions	enable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receive FIFO */
```

```
usart_receive_fifo_enable(USART0);
```

usart_receive_fifo_disable

The description of usart_receive_fifo_disable is shown as below:

Table 3-770. Function usart_receive_fifo_disable

Function name	usart_receive_fifo_disable
Function prototype	void usart_receive_fifo_disable(uint32_t usart_periph);
Function descriptions	disable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receive FIFO */
```

```
usart_receive_fifo_disable(USART0);
```

usart_receive_fifo_counter_number

The description of usart_receive_fifo_counter_number is shown as below:

Table 3-771. Function usart_receive_fifo_counter_number

Function name	usart_receive_fifo_counter_number
Function prototype	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
Function descriptions	read receive FIFO counter number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
uint8_t	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */
```

```
uint8_t temp;
```

```
temp = usart_receive_fifo_counter_number(USART0);
```

usart_flag_get

The description of usart_flag_get is shown as below:

Table 3-772. Function usart_flag_get

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT/CHC/RFCR register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
flag	USART flags, refer to Table 3-703. Enum usart_flag_enum
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise error flag
USART_FLAG_ORER R	overrun error
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_TC	transmission completed
USART_FLAG_TBE	transmit data register empty
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_CTSF	CTS change flag
USART_FLAG_CTS	CTS level
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EB	end of block flag
USART_FLAG_BSY	busy flag
USART_FLAG_AM	address match flag
USART_FLAG_SB	send break flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_WU	wakeup from deep-sleep mode flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFE	receive FIFO empty flag

<i>USART_FLAG_RFF</i>	receive FIFO full flag
<i>USART_FLAG_RFFINT</i>	receive FIFO full interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

usart_flag_clear

The description of usart_flag_clear is shown as below:

Table 3-773. Function usart_flag_clear

Function name	usart_flag_clear
Function prototype	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	clear USART status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
flag	USART flags, refer to Table 3-703. Enum usart_flag_enum
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_NERR</i>	noise detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_IDLE</i>	idle line detected flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_CTSF</i>	CTS change flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_AM</i>	address match flag
<i>USART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>USART_FLAG_EPERR</i>	early parity error flag
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0, USART_FLAG_TC);
```

usart_interrupt_enable

The description of usart_interrupt_enable is shown as below:

Table 3-774. Function usart_interrupt_enable

Function name	usart_interrupt_enable
Function prototype	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	enable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
interrupt	USART interrupt, refer to Table 3-705. Enum usart_interrupt_enum
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	receive FIFO full interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

usart_interrupt_disable

The description of usart_interrupt_disable is shown as below:

Table 3-775. Function usart_interrupt_disable

Function name	usart_interrupt_disable
Function prototype	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	disable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
intterrupt	USART interrupt flag, refer to Table 3-705. Enum usart_interrupt_enum
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	receive FIFO full interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```


usart_interrupt_flag_get

The description of usart_interrupt_flag_get is shown as below:

Table 3-776. Function usart_interrupt_flag_get

Function name	usart_interrupt_flag_get
Function prototype	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	get USART interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-704. Enum usart_interrupt_flag_enum
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_A M	address match interrupt and flag
USART_INT_FLAG_PE RR	parity error interrupt and flag
USART_INT_FLAG_TB E	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RB NE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_ID LE	IDLE line detected interrupt and flag
USART_INT_FLAG_LB D	LIN break detected interrupt and flag
USART_INT_FLAG_W U	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CT S	CTS interrupt and flag
USART_INT_FLAG_ER R_NERR	error interrupt and noise error flag
USART_INT_FLAG_ER R_ORERR	error interrupt and overrun error
USART_INT_FLAG_ER	error interrupt and frame error flag

<i>R_FERR</i>	
<i>USART_INT_FLAG_RFRF</i>	receive FIFO full interrupt and flag
<i>F</i>	
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

usart_interrupt_flag_clear

The description of usart_interrupt_flag_clear is shown as below:

Table 3-777. Function usart_interrupt_flag_clear

Function name	usart_interrupt_flag_clear
Function prototype	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	clear USART interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-704. Enum usart_interrupt_flag_enum
<i>USART_INT_FLAG_PERR</i>	parity error flag
<i>USART_INT_FLAG_ERFERR</i>	frame error flag
<i>USART_INT_FLAG_ERNERR</i>	noise detected flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ERORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_IDLE</i>	idle line detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete flag

USART_INT_FLAG_LB D	LIN break detected flag
USART_INT_FLAG_CTS	CTS change flag
USART_INT_FLAG_RT	receiver timeout flag
USART_INT_FLAG_EB	end of block flag
USART_INT_FLAG_AM	address match flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode flag
USART_INT_FLAG_RF F	receive FIFO full interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.25. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.25.1](#), the WWDGT firmware functions are introduced in chapter [3.25.2](#).

3.25.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

Table 3-778. WWDGT Registers

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

3.25.2. Descriptions of Peripheral functions

WWDGT firmware functions are listed in the table shown as below:

Table 3-779. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the WWDGT configuration
wwdgt_enable	start the WWDGT counter
wwdgt_counter_update	configure the WWDGT counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

wwdgt_deinit

The description of wwdgt_deinit is shown as below:

Table 3-780. Function wwdgt_deinit

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the WWDGT configuration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit();
```

wwdgt_enable

The description of wwdgt_enable is shown as below:

Table 3-781. Function wwdgt_enable

Function name	wwdgt_enable
Function prototype	void wwdgt_enable(void);
Function descriptions	start the WWDGT counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start the WWDGT counter */
```

```
wwdgt_enable();
```

wwdgt_counter_update

The description of wwdgt_counter_update is shown as below:

Table 3-782. Function wwdgt_counter_update

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the WWDGT value
Precondition	-
The called functions	-
Input parameter{in}	
counter_value	0x00 - 0x7F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(0x7F);
```

wwdgt_config

The description of wwdgt_config is shown as below:

Table 3-783. Function wwdgt_config

Function name	wwdgt_config
Function prototype	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
Function descriptions	configure counter value, window value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
counter	0x00 - 0x7F
Input parameter{in}	

window	0x00 - 0x7F
Input parameter{in}	
prescaler	wwdgt prescaler value
<i>WWDGT_CFG_PSC_D</i> <i>IV1</i>	the time base of window watchdog counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_D</i> <i>IV2</i>	the time base of window watchdog counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_D</i> <i>IV4</i>	the time base of window watchdog counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_D</i> <i>IV8</i>	the time base of window watchdog counter = (PCLK1/4096)/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* confiure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

wwdgt_interrupt_enable

The description of wwdgt_interrupt_enable is shown as below:

Table 3-784. Function wwdgt_interrupt_enable

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

wwdgt_flag_get

The description of wwdgt_flag_get is shown as below:

Table 3-785. Function wwdgt_flag_get

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

wwdgt_flag_clear

The description of wwdgt_flag_clear is shown as below:

Table 3-786. Function wwdgt_flag_clear

Function name	wwdgt_flag_clear
Function prototype	void wwdgt_flag_clear(void);
Function descriptions	clear early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Jul.20, 2023
1.1	Add SES project relevant descriptions	Jan.16, 2024
1.2	<p>1. Modify the parameter of i2c_transfer_byte_number_config in <u>3.14.</u></p> <p style="text-align: center;"><u>I2C.</u></p> <p>2. Remove the void i2c_nack_disable(uint32_t i2c_periph) interface in <u>3.14.</u> <u>I2C.</u></p>	Aug.9, 2025
1.3	<p>1. Add parameter of API fwdgt_config in <u>3.11.FWDGT.</u></p> <p>2. Add qspi_data_length_config API in <u>3.17.</u></p> <p style="text-align: center;"><u>QSPI.</u></p> <p>3. Add timer_channel_input_remap_config API in <u>3.22.</u> <u>TIMER.</u></p>	Feb.4, 2026

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.